**Mysql**

## 一、Mysql概述

**1 Mysql是最流行的关系型数据库管理，在Web应用方面MySQL是最好的RDBMS**：关系数据库管理系统

**2 数据库？**

数据库（Database）是按照数据结构来组织，存储和管理数据的仓库。每个数据库都有一个或多个不同的（应用程序编程接口）API用于创建、访问、管理、搜索和复制所保存的数据（我们也可以将数据存储在文件中，但是文件的读取速度相对较慢）所以使用关系型数据库管理系统，建立在关系模型基础上的数据库，借助于集合代数等数学概念和方法来处理数据库中的数据

**3 数据库特点**

@数据以表格的形式出现

@每行为各种记录名称

@每列为记录名称所对应的数据域

@许多的行和列组成一张表单

@若干的表单组成database

**4 RDBMS术语**

数据库：数据库是一些关联表的集合

数据表：表是数据的矩阵

列：相同的数据

行：一组相关的数据

冗余：存储两倍数据，冗余可以使系统的速度更快

主键：唯一，利用主键来查询数据

外键：用于关联两个表

复合键：将多个列作为一个索引键，一般用于复合索引

**5 MySQL数据库**

关联数据库将数据保存在不同的表中，而不是将所有的数据放在一个大仓库内，这样就增加了速度并提高了灵活性，MySQL是开源的，支持大型的数据库，使用标准的SQL数据语言形式，采用了GPL协议，可以定制，修改源码，制定自己的MySQL系统

## 二、Mysql常用命令

**1 基本命令**

（1）连接Mysql

格式： mysql -h主机地址 -u用户名 －p用户密码

（2）连接到本机上的MYSQL。

首先打开DOS窗口，然后进入目录mysql\bin，再键入命令mysql -u root -p，回车后提示你输密码.注意用户名前可以有空格也可以没有空格，但是密码前必须没有空格，否则让你重新输入密码。

如果刚安装好MYSQL，超级用户root是没有密码的，故直接回车即可进入到MYSQL中了，MYSQL的提示符是： mysql>

连接到远程主机上的MYSQL。假设远程主机的IP为：110.110.110.110，用户名为root,密码为abcd123。则键入以下命令：

mysql -h110.110.110.110 -u root -p 123;（注:u与root之间可以不用加空格，其它也一样）

退出MYSQL命令： exit （回车）

（3）修改密码

格式：mysqladmin -u用户名 -p旧密码 password 新密码

给root加个密码ab12。

首先在DOS下进入目录mysql\bin，然后键入以下命令

mysqladmin -u root -password ab12

注：因为开始时root没有密码，所以-p旧密码一项就可以省略了。

再将root的密码改为djg345。

mysqladmin -u root -p ab12 password djg345

（4）增加新用户

注意：和上面不同，下面的因为是MYSQL环境中的命令，所以后面都带一个分号作为命令结束符

格式：grant select on 数据库.\* to 用户名@登录主机 identified by “密码”

1、增加一个用户test1密码为abc，让他可以在任何主机上登录，并对所有数据库有查询、插入、修改、删除的权限。首先用root用户连入MYSQL，然后键入以下命令：

grant select,insert,update,delete on \*.\* to [email=test1@”%]test1@”%[/email]” Identified by “abc”;

但增加的用户是十分危险的，你想如某个人知道test1的密码，那么他就可以在internet上的任何一台电脑上登录你的mysql数据库并对你的数据可以为所欲为了，解决办法见2。

2、增加一个用户test2密码为abc,让他只可以在localhost上登录，并可以对数据库mydb进行查询、插入、修改、删除的操作（localhost指本地主机，即MYSQL数据库所在的那台主机），这样用户即使用知道test2的密码，他也无法从internet上直接访问数据库，只能通过MYSQL主机上的web页来访问了。

grant select,insert,update,delete on mydb.\* to [email=test2@localhost]test2@localhost[/email] identified by “abc”;

如果你不想test2有密码，可以再打一个命令将密码消掉。

grant select,insert,update,delete on mydb.\* to [email=test2@localhost]test2@localhost[/email] identified by “”;

**2 创建数据库**

注意：创建数据库之前要先连接Mysql服务器

命令：create database <数据库名>

例1：建立一个名为xhkdb的数据库

mysql> create database xhkdb;

例2：创建数据库并分配用户

①CREATE DATABASE 数据库名;

②GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP,ALTER ON 数据库名.\* TO 数据库名@localhost IDENTIFIED BY '密码';

③SET PASSWORD FOR '数据库名'@'localhost' = OLD\_PASSWORD('密码');

依次执行3个命令完成数据库创建。注意：中文 “密码”和“数据库”是户自己需要设置的。

**3 显示数据库**

命令：show databases （注意：最后有个s）

mysql> show databases;

注意：为了不再显示的时候乱码，要修改数据库默认编码。以下以GBK编码页面为例进行说明：

1、修改MYSQL的配置文件：my.ini里面修改default-character-set=gbk

2、代码运行时修改：

①Java代码：jdbc:mysql://localhost:3306/test?useUnicode=true&characterEncoding=gbk

②PHP代码：header("Content-Type:text/html;charset=gb2312");

③C语言代码：int mysql\_set\_character\_set( MYSQL \* mysql, char \* csname)；

该函数用于为当前连接设置默认的字符集。字符串csname指定了1个有效的字符集名称。连接校对成为字符集的默认校对。该函数的工作方式与SET NAMES语句类似，但它还能设置mysql- > charset的值，从而影响了由mysql\_real\_escape\_string() 设置的字符集。

**4 删除数据库**

命令：drop database <数据库名>

例如：删除名为 xhkdb的数据库

mysql> drop database xhkdb;

例子1：删除一个已经确定存在的数据库

mysql> drop database drop\_database;

Query OK, 0 rows affected (0.00 sec)

例子2：删除一个不确定存在的数据库

mysql> drop database drop\_database;

ERROR 1008 (HY000): Can't drop database 'drop\_database'; database doesn't exist

//发生错误，不能删除'drop\_database'数据库，该数据库不存在。

mysql> drop database if exists drop\_database;

Query OK, 0 rows affected, 1 warning (0.00 sec)//产生一个警告说明此数据库不存在

mysql> create database drop\_database;

Query OK, 1 row affected (0.00 sec)

mysql> drop database if exists drop\_database;//if exists 判断数据库是否存在，不存在也不产生错误

Query OK, 0 rows affected (0.00 sec)

**5 连接数据库**

命令： use <数据库名>

例如：如果xhkdb数据库存在，尝试存取它：

mysql> use xhkdb;

屏幕提示：Database changed

use 语句可以通告MySQL把db\_name数据库作为默认（当前）数据库使用，用于后续语句。该数据库保持为默认数据库，直到语段的结尾，或者直到发布一个不同的USE语句：

mysql> USE db1;

mysql> SELECT COUNT(\*) FROM mytable; # selects from db1.mytable

mysql> USE db2;

mysql> SELECT COUNT(\*) FROM mytable; # selects from db2.mytable

使用USE语句为一个特定的当前的数据库做标记，不会阻碍您访问其它数据库中的表。下面的例子可以从db1数据库访问作者表，并从db2数据库访问编辑表：

mysql> USE db1;

mysql> SELECT author\_name,editor\_name FROM author,db2.editor

-> WHERE author.editor\_id = db2.editor.editor\_id;

USE语句被设立出来，用于与Sybase相兼容。

有些网友问到，连接以后怎么退出。其实，不用退出来，use 数据库后，使用show databases就能查询所有数据库，如果想跳到其他数据库，用

use 其他数据库名字

就可以了。

**6 当前选择的数据库**

命令：mysql> select database();

**7 显示MYSQL的版本**

mysql> select version();

+-----------------------+

| version() |

+-----------------------+

| 6.0.4-alpha-community |

+-----------------------+

1 row in set (0.02 sec)

**8 显示当前时间**

mysql> select now();

+---------------------+

| now() |

+---------------------+

| 2009-09-15 22:35:32 |

+---------------------+

1 row in set (0.04 sec)

**9 显示年月日**

SELECT DAYOFMONTH(CURRENT\_DATE);

+--------------------------+

| DAYOFMONTH(CURRENT\_DATE) |

+--------------------------+

| 15 |

+--------------------------+

1 row in set (0.01 sec)

SELECT MONTH(CURRENT\_DATE);

+---------------------+

| MONTH(CURRENT\_DATE) |

+---------------------+

| 9 |

+---------------------+

1 row in set (0.00 sec)

SELECT YEAR(CURRENT\_DATE);

+--------------------+

| YEAR(CURRENT\_DATE) |

+--------------------+

| 2009 |

+--------------------+

1 row in set (0.00 sec)

**10 显示字符串**

mysql> SELECT "welecome to my blog!";

+----------------------+

| welecome to my blog! |

+----------------------+

| welecome to my blog! |

+----------------------+

1 row in set (0.00 sec)

**11 当计算器用**

select ((4 \* 4) / 10 ) + 25;

+----------------------+

| ((4 \* 4) / 10 ) + 25 |

+----------------------+

| 26.60 |

+----------------------+

1 row in set (0.00 sec)

**12 串接字符串**

select CONCAT(f\_name, " ", l\_name)

AS Name

from employee\_data

where title = 'Marketing Executive';

+---------------+

| Name |

+---------------+

| Monica Sehgal |

| Hal Simlai |

| Joseph Irvine |

+---------------+

3 rows in set (0.00 sec)

注意：这里用到CONCAT()函数，用来把字符串串接起来。另外，我们还用到以前学到的AS给结果列'CONCAT(f\_name, " ", l\_name)'起了个假名。

**13 创建数据表**

命令：create table <表名> ( <字段名1> <类型1> [,..<字段名n> <类型n>]);

例如，建立一个名为MyClass的表，

字段名 数字类型 数据宽度 是否为空 是否主键 自动增加 默认值

id int 4 否 primary key auto\_increment

name char 20 否

sex int 4 否 0

degree double 16 是

mysql> create table MyClass(

> id int(4) not null primary key auto\_increment,

> name char(20) not null,

> sex int(4) not null default '0',

> degree double(16,2));

**14 删除数据表**

命令：drop table <表名>

例如：删除表名为 MyClas 的表

mysql> drop table MyClass;

DROP TABLE用于取消一个或多个表。您必须有每个表的DROP权限。所有的表数据和表定义会被取消，所以使用本语句要小心！

注意：对于一个带分区的表，DROP TABLE会永久性地取消表定义，取消各分区，并取消储存在这些分区中的所有数据。DROP TABLE还会取消与被取消的表有关联的分区定义（.par）文件。

对与不存在的表，使用IF EXISTS用于防止错误发生。当使用IF EXISTS时，对于每个不存在的表，会生成一个NOTE。

RESTRICT和CASCADE可以使分区更容易。目前，RESTRICT和CASCADE不起作用。

**15 表插入数据**

命令：insert into <表名> [( <字段名1>[,..<字段名n > ])] values ( 值1 )[, ( 值n )]

例如：往表 MyClass中插入二条记录, 这二条记录表示：编号为1的名为Tom的成绩为96.45, 编号为2 的名为Joan 的成绩为82.99， 编号为3 的名为Wang 的成绩为96.5。

mysql> insert into MyClass values(1,'Tom',96.45),(2,'Joan',82.99), (2,'Wang', 96.59);

注意：insert into每次只能向表中插入一条记录。

**16 查询表中的数据**

1)、查询所有行

命令： select <字段1，字段2，...> from < 表名 > where < 表达式 >

例如：查看表 MyClass 中所有数据

mysql> select \* from MyClass;

2）、查询前几行数据

例如：查看表 MyClass 中前2行数据

mysql> select \* from MyClass order by id limit 0,2;

select一般配合where使用，以查询更精确更复杂的数据。

**17 删除表中数据**

命令：delete from 表名 where 表达式

例如：删除表 MyClass中编号为1 的记录

mysql> delete from MyClass where id=1;

下面是一个删除数据前后表的对比。

FirstName LastName Age

Peter Griffin 35

Glenn Quagmire 33

下面以PHP代码为例删除 "Persons" 表中所有 LastName='Griffin' 的记录：

<?php

$con = mysql\_connect("localhost","peter","abc123");

if (!$con)

{

die('Could not connect: ' . mysql\_error());

}

mysql\_select\_db("my\_db", $con);

mysql\_query("DELETE FROM Persons WHERE LastName='Griffin'"); mysql\_close($con);

?>

在这次删除之后，表是这样的：

FirstName LastName Age

Glenn Quagmire 33

**18 修改表中数据**

语法：update 表名 set 字段=新值,… where 条件

mysql> update MyClass set name='Mary' where id=1;

例子1：单表的MySQL UPDATE语句：

UPDATE [LOW\_PRIORITY] [IGNORE] tbl\_name SET col\_name1=expr1 [, col\_name2=expr2 ...] [WHERE where\_definition] [ORDER BY ...] [LIMIT row\_count]

例子2：多表的UPDATE语句：

UPDATE [LOW\_PRIORITY] [IGNORE] table\_references SET col\_name1=expr1 [, col\_name2=expr2 ...] [WHERE where\_definition]

UPDATE语法可以用新值更新原有表行中的各列。SET子句指示要修改哪些列和要给予哪些值。WHERE子句指定应更新哪些行。如果没有WHERE子句，则更新所有的行。如果指定了ORDER BY子句，则按照被指定的顺序对行进行更新。LIMIT子句用于给定一个限值，限制可以被更新的行的数目。

**19 DDL语句**

@增加字段

命令：alter table 表名 add字段 类型 其他;

例如：在表MyClass中添加了一个字段passtest，类型为int(4)，默认值为0

mysql> alter table MyClass add passtest int(4) default '0'

@加索引

mysql> alter table 表名 add index 索引名 (字段名1[，字段名2 …]);

例子： mysql> alter table employee add index emp\_name (name);

@加主关键字的索引

mysql> alter table 表名 add primary key (字段名);

例子： mysql> alter table employee add primary key(id);

@加唯一限制条件的索引

mysql> alter table 表名 add unique 索引名 (字段名);

例子： mysql> alter table employee add unique emp\_name2(cardnumber);

@删除某个索引

mysql> alter table 表名 drop index 索引名;

例子： mysql>alter table employee drop index emp\_name;

@增加字段：

mysql> ALTER TABLE table\_name ADD field\_name field\_type;

@修改原字段名称及类型：

mysql> ALTER TABLE table\_name CHANGE old\_field\_name new\_field\_name field\_type;

@删除字段：

MySQL ALTER TABLE table\_name DROP field\_name;

**20 修改表名**

命令：rename table 原表名 to 新表名;

例如：在表MyClass名字更改为YouClass

mysql> rename table MyClass to YouClass;

当你执行 RENAME 时，你不能有任何锁定的表或活动的事务。你同样也必须有对原初表的 ALTER 和 DROP 权限，以及对新表的 CREATE 和 INSERT 权限。

如果在多表更名中，MySQL 遭遇到任何错误，它将对所有被更名的表进行倒退更名，将每件事物退回到最初状态。

RENAME TABLE 在 MySQL 3.23.23 中被加入。

**21 备份数据库**

命令在DOS的[url=file://\\mysql\\bin]\\mysql\\bin[/url]目录下执行

（1）导出整个数据库

导出文件默认是存在mysql\bin目录下

mysqldump -u 用户名 -p 数据库名 > 导出的文件名

mysqldump -u user\_name -p123456 database\_name > outfile\_name.sql

（2）导出一个表

mysqldump -u 用户名 -p 数据库名 表名> 导出的文件名

mysqldump -u user\_name -p database\_name table\_name > outfile\_name.sql

（3）导出一个数据库结构

mysqldump -u user\_name -p -d –add-drop-table database\_name > outfile\_name.sql

-d 没有数据 –add-drop-table 在每个create语句之前增加一个drop table

（4）带语言参数导出

mysqldump -uroot -p –default-character-set=latin1 –set-charset=gbk –skip-opt database\_name > outfile\_name.sql

例如将aaa库备份到文件back\_aaa中：

　　[root@test1 root]# cd　/home/data/mysql

　　[root@test1 mysql]# mysqldump -u root -p --opt aaa > back\_aaa

**22 一个建库和建表的实例1**

drop database if exists school; //如果存在SCHOOL则删除

create database school; //建立库SCHOOL

use school; //打开库SCHOOL

create table teacher //建立表TEACHER

(

id int(3) auto\_increment not null primary key,

name char(10) not null,

address varchar(50) default ‘深圳’,

year date

); //建表结束

//以下为插入字段

insert into teacher values(”,’allen’,'大连一中’,'1976-10-10′);

insert into teacher values(”,’jack’,'大连二中’,'1975-12-23′);

如果你在mysql提示符键入上面的命令也可以，但不方便调试。

（1）你可以将以上命令原样写入一个文本文件中，假设为school.sql，然后复制到c:\\下，并在DOS状态进入目录[url=file://\\mysql\\bin]\\mysql\\bin[/url]，然后键入以下命令：

mysql -uroot -p密码 < c:\\school.sql

如果成功，空出一行无任何显示；如有错误，会有提示。（以上命令已经调试，你只要将//的注释去掉即可使用）。

（2）或者进入命令行后使用 mysql> source c:\\school.sql; 也可以将school.sql文件导入数据库中。

**23 一个建库和建表的实例**

drop database if exists school; //如果存在SCHOOL则删除

create database school; //建立库SCHOOL

use school; //打开库SCHOOL

create table teacher //建立表TEACHER

(

id int(3) auto\_increment not null primary key,

name char(10) not null,

address varchar(50) default ''深圳'',

year date

); //建表结束

//以下为插入字段

insert into teacher values('''',''glchengang'',''深圳一中'',''1976-10-10'');

insert into teacher values('''',''jack'',''深圳一中'',''1975-12-23'');

注：在建表中

1、将ID设为长度为3的数字字段:int(3)；并让它每个记录自动加一:auto\_increment；并不能为空:not null；而且让他成为主字段primary key。

2、将NAME设为长度为10的字符字段

3、将ADDRESS设为长度50的字符字段，而且缺省值为深圳。

4、将YEAR设为日期字段。

## 三、数据类型

**1 MySQL的数据类型**

主要包括以下五大类：

整数类型：BIT、BOOL、TINY INT、SMALL INT、MEDIUM INT、 INT、 BIG INT

浮点数类型：FLOAT、DOUBLE、DECIMAL

字符串类型：CHAR、VARCHAR、TINY TEXT、TEXT、MEDIUM TEXT、LONGTEXT、TINY BLOB、BLOB、MEDIUM BLOB、LONG BLOB

日期类型：Date、DateTime、TimeStamp、Time、Year

其他数据类型：BINARY、VARBINARY、ENUM、SET、Geometry、Point、MultiPoint、LineString、MultiLineString、Polygon、GeometryCollection等

date “1000-01-01”到“9999-12-31” 3字节

time “-838:59:59”到“838:59:59” 3字节

datetime “1000-01-01 00:00:00” 到“9999-12-31 23:59:59” 8字节

timestamp 19700101000000 到2037 年的某个时刻 4字节

year 1901 到2155 1字节

**2 整型**

MySQL数据类型 含义（有符号）

tinyint(m) 1个字节  范围(-128~127)

smallint(m) 2个字节  范围(-32768~32767)

mediumint(m) 3个字节  范围(-8388608~8388607)

int(m) 4个字节  范围(-2147483648~2147483647)

bigint(m) 8个字节  范围(+-9.22\*10的18次方)

取值范围如果加了unsigned，则最大值翻倍，如tinyint unsigned的取值范围为(0~256)。

int(m)里的m是表示SELECT查询结果集中的显示宽度，并不影响实际的取值范围，没有影响到显示的宽度，不知道这个m有什么用。

**3 浮点型(float和doubl**e)

MySQL数据类型 含义

float(m,d) 单精度浮点型    8位精度(4字节)     m总个数，d小数位

double(m,d) 双精度浮点型    16位精度(8字节)    m总个数，d小数位

设一个字段定义为float(6,3)，如果插入一个数123.45678,实际数据库里存的是123.457，但总个数还以实际为准，即6位。整数部分最大是3位，如果插入数12.123456，存储的是12.1234，如果插入12.12，存储的是12.1200.

**4 定点数**

浮点型在数据库中存放的是近似值，而定点类型在数据库中存放的是精确值。

decimal(m,d) 参数m<65 是总个数，d<30且 d<m 是小数位。

**5 字符串(char,varchar,\_text)**

MySQL数据类型 含义

char(n) 固定长度，最多255个字符

varchar(n) **变长**长度，最多65535个字符

tinytext 可变长度，最多255个字符

text 可变长度，最多65535个字符

mediumtext 可变长度，最多2的24次方-1个字符

longtext 可变长度，最多2的32次方-1个字符

**char定义的是固定长度，长度范围为0-255，存储时，如果字符数没有达到定义的位数，会在后面用空格补全存入数据库中，在上例中，name实际存储在数据中的数据为'zejin '**

**varchar是变长长度，长度范围为0-65535，存储时，如果字符没有达到定义的位数，也不会在后面补空格，在上例subject字段中，实际存储在数据中的数据为'zejin '，当然还有一或两个字节来描述该字节长度**

**1）text和varchar基本相同**

**2）text会忽略指定的大小这和varchar有所不同，text不能有默认值**

**3）text尾部有空格不会被截断**

**4）text使用额 外的2个字节来存储数据的大小，varchar根据存储数据的大小选择用几个字节来存储**

**5）text的65535字节全部用来存储数据，varchar则会 占用1－3个字节去存储数据大小**

**6 char和varchar：**

（1）char(n) 若存入字符数小于n，则以空格补于其后，查询之时再将空格去掉。所以char类型存储的字符串末尾不能有空格，varchar不限于此。

（2）char(n) 固定长度，char(4)不管是存入几个字符，都将占用4个字节，varchar是存入的实际字符数+1个字节（n<=255）或2个字节(n>255)，所以varchar(4),存入3个字符将占用4个字节。

（3）char类型的字符串检索速度要比varchar类型的快。

**7 varchar和text**

（1）varchar可指定n，text不能指定，内部存储varchar是存入的实际字符数+1个字节（n<=255）或2个字节(n>255)，text是实际字符数+2个字

节。

（2）text类型不能有默认值。

（3）varchar可直接创建索引，text创建索引要指定前多少个字符。varchar查询速度快于text,在都创建索引的情况下，text的索引似乎不起作用。

**8 二进制数据(\_Blob)**

1.\_BLOB和\_text存储方式不同，\_TEXT以文本方式存储，英文存储区分大小写，而\_Blob是以二进制方式存储，不分大小写。

2.\_BLOB存储的数据只能整体读出。

3.\_TEXT可以指定字符集，\_BLO不用指定字符集。

**9 日期时间类型**

MySQL数据类型 含义

date 日期 '2008-12-2'

time 时间 '12:25:36'

datetime 日期时间 '2008-12-2 22:06:44'

timestamp 自动存储记录修改时间

若定义一个字段为timestamp，这个字段里的时间数据会随其他字段修改的时候自动刷新，所以这个数据类型的字段可以存放这条记录最后被修改的时间。

**10 数据类型的属性**

**MySQL关键字 含义**

NULL 数据列可包含NULL值

NOT NULL 数据列不允许包含NULL值

DEFAULT 默认值

PRIMARY KEY 主键

AUTO\_INCREMENT 自动递增，适用于整数类型

UNSIGNED 无符号

CHARACTER SET name 指定一个字符集

**11 MYSQL数据类型的长度和范围**

各数据类型及字节长度一览表：

**数据类型 字节长度 范围或用法**

Bit 1 无符号[0,255]，有符号[-128,127]，天缘博客备注：BIT和BOOL布尔型都占用1字节

TinyInt 1 整数[0,255]

SmallInt 2 无符号[0,65535]，有符号[-32768,32767]

MediumInt 3 无符号[0,2^24-1]，有符号[-2^23,2^23-1]]

Int 4 无符号[0,2^32-1]，有符号[-2^31,2^31-1]

BigInt 8 无符号[0,2^64-1]，有符号[-2^63 ,2^63 -1]

Float(M,D) 4 单精度浮点数。天缘博客提醒这里的D是精度，如果D<=24则为默认的FLOAT，如果D>24则会自动被转换为DOUBLE型。

Double(M,D) 8  双精度浮点。

Decimal(M,D) M+1或M+2 未打包的浮点数，用法类似于FLOAT和DOUBLE，天缘博客提醒您如果在ASP中使用到Decimal数据类型，直接从数据库读出来的Decimal可能需要先转换成Float或Double类型后再进行运算。

Date 3 以YYYY-MM-DD的格式显示，比如：2009-07-19

Date Time 8 以YYYY-MM-DD HH:MM:SS的格式显示，比如：2009-07-19 11：22：30

TimeStamp 4 以YYYY-MM-DD的格式显示，比如：2009-07-19

Time 3 以HH:MM:SS的格式显示。比如：11：22：30

Year 1 以YYYY的格式显示。比如：2009

Char(M) M 定长字符串。

VarChar(M) M 变长字符串，要求M<=255

Binary(M) M 类似Char的二进制存储，特点是插入定长不足补0

VarBinary(M) M 类似VarChar的变长二进制存储，特点是定长不补0

Tiny Text Max:255 大小写不敏感

Text Max:64K 大小写不敏感

Medium Text Max:16M 大小写不敏感

Long Text Max:4G 大小写不敏感

TinyBlob Max:255 大小写敏感

Blob Max:64K 大小写敏感

MediumBlob Max:16M 大小写敏感

LongBlob Max:4G 大小写敏感

Enum 1或2 最大可达65535个不同的枚举值

Set 可达8 最大可达64个不同的值

## 四、常用运算符

**1.算数运算符**

加

mysql> select 1+2;

减

mysql> select 2-1;

乘

mysql> select 2\*3;

除

mysql> select 5/3;

商

mysql> SELECT 5 DIV 2;

模

mysql> select 5%2,mod(5,2);

**2.比较运算符**

等于

mysql> select 1=0,1=1,null=null;

不等于

mysql> select 1<>0,1<>1,null<>null;

安全等于

mysql> select 1<=>1,2<=>0,0<=>0,null<=>null;

小于

mysql> select 'a'<'b','a'<'a','a'<'c',1<2;

小于等于

mysql> select 'bdf'<='b','b'<='b',0<1;

大于

mysql> select 'a'>'b','abc'>'a',1>0;

大于等于

mysql> select 'a'>='b','abc'>='a',1>=0,1>=1;

BETWEEN

mysql> select 10 between 10 and 20, 9 between 10 and 20;

IN

mysql> select 1 in (1,2,3), 't' in ('t','a','b','l','e'), 0 in (1,2);

IS NULL

mysql> select 0 is null,null is null;

IS NOT NULL

mysql> select 0 is not null, null is not null;

LIKE

mysql> select 123456 like '123%', 123456 like '%123%', 123456 like '%321%';

REGEXP

mysql> select 'abcdef' regexp 'ab', 'abcdefg' regexp 'k';

**3.逻辑运算符**

非

mysql> select not 0, not 1, not null;

mysql> select ! 0, ! 1, ! null;

与

mysql> select (1 and 1), (0 and 1), (3 and 1), (1 and null);

mysql> select (1 && 1), (0 && 1), (3 && 1), (1 && null);

或

mysql> select (1 or 0), (0 or 0), (1 or null), (1 or 1), (null or null);

mysql> select (1 || 0), (0 || 0), (1 || null), (1 || 1), (null || null);

异或

mysql> select (1 xor 1), (0 xor 0), (1 xor 0), (0 xor 1), (null xor 1);

mysql> select (1 ^ 1), (0 ^ 0), (1 ^ 0), (0 ^ 1), (null ^ 1);

**4.位运算符**

位与

mysql> select 2&3;

mysql> select 2&3&4;

位或

mysql> select 2|3;

位异或

mysql> select 2^3;

位取反

mysql> select ~1,~18446744073709551614;

位右移

mysql> select 100>>3;

位左移

mysql> select 100<<3;

**5.运算符优先级顺序**

最高优先级 :=

1 ||, OR, XOR

2 &&, AND

3 BETWEEN, CASE, WHEN, THEN, ELSE

4 =, <=>, >=, >, <=, <, <>, !=, IS, LIKE, REGEXP, IN

5 |

6 &

7 <<, >>

8 -, +

9 \*, /, DIV, %, MOD

10 ^

11 - (unary minus), ~ (unary bit inversion)

12 !, NOT

最低优先级 BINARY, COLLATE

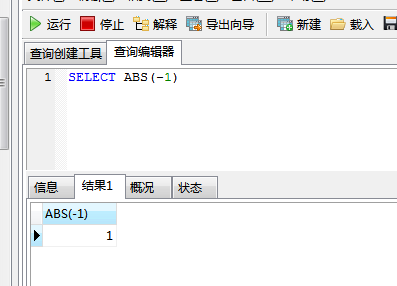
## 五、SQL函数

**1 数学函数-数学函数主要用于处理数字，包括整型、浮点数等。**

ABS(x)

返回x的绝对值

SELECT ABS(-1) -- 返回1



CEIL(x),CEILING(x)

返回大于或等于x的最小整数

SELECT CEIL(1.5) -- 返回2

FLOOR(x)

返回小于或等于x的最大整数

SELECT FLOOR(1.5) -- 返回1

RAND()

返回0->1的随机数

SELECT RAND() --0.93099315644334

RAND(x)

返回0->1的随机数，x值相同时返回的随机数相同

SELECT RAND(2) --1.5865798029924

SIGN(x)

返回x的符号，x是负数、0、正数分别返回-1、0和1

SELECT SIGN(-10) -- (-1)

PI()

返回圆周率(3.141593）

SELECT PI() --3.141593

TRUNCATE(x,y)

返回数值x保留到小数点后y位的值（与ROUND最大的区别是不会进行四舍五入）

SELECT TRUNCATE(1.23456,3) -- 1.234

ROUND(x) 返回离x最近的整数　　SELECT ROUND(1.23456) --1

ROUND(x,y)

保留x小数点后y位的值，但截断时要进行四舍五入

SELECT ROUND(1.23456,3) -- 1.235

POW(x,y).POWER(x,y)

返回x的y次方

SELECT POW(2,3) -- 8

SQRT(x)

返回x的平方根

SELECT SQRT(25) -- 5

EXP(x)

返回e的x次方

SELECT EXP(3) -- 20.085536923188

MOD(x,y)

返回x除以y以后的余数

SELECT MOD(5,2) -- 1

LOG(x)

返回自然对数(以e为底的对数)

SELECT LOG(20.085536923188) -- 3

LOG10(x)

返回以10为底的对数

SELECT LOG10(100) -- 2

RADIANS(x)

将角度转换为弧度

SELECT RADIANS(180) -- 3.1415926535898

SIN(x)

求正弦值(参数是弧度)

SELECT SIN(RADIANS(30)) -- 0.5

ASIN(x) 求反正弦值(参数是弧度)

COS(x) 求余弦值(参数是弧度)

ACOS(x) 求反余弦值(参数是弧度)

TAN(x) 求正切值(参数是弧度)

ATAN(x) ATAN2(x) 求反正切值(参数是弧度)

COT(x) 求余切值(参数是弧度)

**2 字符串函数-字符串函数是MySQL中最常用的一类函数，字符串函数主要用于处理表中的字符串。**

CHAR\_LENGTH(s)

返回字符串s的字符数

SELECT CHAR\_LENGTH('你好123') -- 5

LENGTH(s)

返回字符串s的长度

SELECT LENGTH('你好123') -- 9

CONCAT(s1,s2,...)

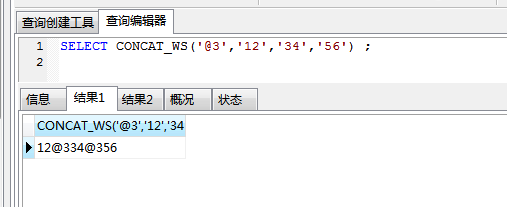
将字符串s1,s2等多个字符串合并为一个字符串

SELECT CONCAT('12','34') -- 1234

CONCAT\_WS(x,s1,s2,...)

同CONCAT(s1,s2,...)函数，但是每个字符串直接要加上x

SELECT CONCAT\_WS('@','12','34') -- 12@34



INSERT(s1,x,len,s2)

将字符串s2替换s1的x位开始长度为len的字符串

SELECT INSERT('12345',1,3,'abc') -- abc45

UPPER(s),UCAASE(S)

将字符串s的所有字母变成写字母

SELECT UPPER('abc') -- ABC

LOWER(s),LCASE(s)

将字符串s的所有字母变成小写字母

SELECT LOWER('ABC') -- abc

LEFT(s,n)

返回字符串s的前n个字符

SELECT LEFT('abcde',2) -- ab

RIGHT(s,n)

返回字符串s的后n个字符

SELECT RIGHT('abcde',2) -- de

LPAD(s1,len,s2)

字符串s2来填充s1的开始处，使字符串长度达到len

SELECT LPAD('abc',5,'xx') -- xxabc

RPAD(s1,len,s2)

字符串s2来填充s1的结尾处，使字符串的长度达到len

SELECT RPAD('abc',5,'xx') -- abcxx

LTRIM(s) 去掉字符串s开始处的空格

RTRIM(s) 去掉字符串s结尾处的空格

TRIM(s) 去掉字符串s开始和结尾处的空格

TRIM(s1 FROM s)

去掉字符串s中开始处和结尾处的字符串s1

SELECT TRIM('@' FROM '@@abc@@') -- abc

REPEAT(s,n)

将字符串s重复n次

SELECT REPEAT('ab',3) -- ababab

SPACE(n) 返回n个空格

REPLACE(s,s1,s2) 将字符串s2替代字符串s中的字符串s1

SELECT REPLACE('abc','a','x') --xbc

STRCMP(s1,s2) 比较字符串s1和s2

SUBSTRING(s,n,len) 获取从字符串s中的第n个位置开始长度为len的字符串

MID(s,n,len) 同SUBSTRING(s,n,len)

LOCATE(s1,s),POSITION(s1 IN s)

从字符串s中获取s1的开始位置

SELECT LOCATE('b', 'abc') -- 2

INSTR(s,s1)

从字符串s中获取s1的开始位置

SELECT INSTR('abc','b') -- 2

REVERSE(s)

将字符串s的顺序反过来

SELECT REVERSE('abc') -- cba

ELT(n,s1,s2,...)

返回第n个字符串

SELECT ELT(2,'a','b','c') -- b

EXPORT\_SET(x,s1,s2)

返回一个字符串，在这里对于在“bits”中设定每一位，你得到一个“on”字符串，并且对于每个复位(reset)的位，你得到一个 “off”字符串。每个字符串用“separator”分隔(缺省“,”)，并且只有“bits”的“number\_of\_bits” (缺省64)位被使用。

SELECT EXPORT\_SET(5,'Y','N',',',4) -- Y,N,Y,N

FIELD(s,s1,s2...)

返回第一个与字符串s匹配的字符串位置

SELECT FIELD('c','a','b','c') -- 3

FIND\_IN\_SET(s1,s2) 返回在字符串s2中与s1匹配的字符串的位置

MAKE\_SET(x,s1,s2) 返回一个集合 (包含由“,”

字符分隔的子串组成的一个 字符串)，由相应的位在bits集合中的的字符串组成。str1对应于位0，str2对 应位1，等等。

SELECT MAKE\_SET(1|4,'a','b','c'); -- a,c

SUBSTRING\_INDEX

返回从字符串str的第count个出现的分隔符delim之后的子串。

如果count是正数，返回第count个字符左边的字符串。

如果count是负数，返回第(count的绝对值(从右边数))个字符右边的字符串。

SELECT SUBSTRING\_INDEX('a\*b','\*',1) -- a

SELECT SUBSTRING\_INDEX('a\*b','\*',-1) -- b

SELECT SUBSTRING\_INDEX(SUBSTRING\_INDEX('a\*b\*c\*d\*e','\*',3),'\*',-1) -- c

LOAD\_FILE(file\_name)

读入文件并且作为一个字符串返回文件内容。文件必须在服务器上，你必须指定到文件的完整路径名，而且你必须有file权 限。文件必须所有内容都是可读的并且小于max\_allowed\_packet。 如果文件不存在或由于上面原因之一不能被读出，函数返回NULL。

**3 日期时间函数-MySQL的日期和时间函数主要用于处理日期时间。**

CURDATE(),CURRENT\_DATE

返回当前日期

SELECT CURDATE()

->2014-12-17

CURTIME(),CURRENT\_TIME

返回当前时间

SELECT CURTIME()

->15:59:02

NOW(),CURRENT\_TIMESTAMP(),LOCALTIME(),

SYSDATE(),LOCALTIMESTAMP()

返回当前日期和时间

SELECT NOW()

->2014-12-17 15:59:02

UNIX\_TIMESTAMP()

以UNIX时间戳的形式返回当前时间

SELECT UNIX\_TIMESTAMP()

->1418803177

UNIX\_TIMESTAMP(d)

将时间d以UNIX时间戳的形式返回

SELECT UNIX\_TIMESTAMP('2011-11-11 11:11:11')

->1320981071

FROM\_UNIXTIME(d)

将UNIX时间戳的时间转换为普通格式的时间

SELECT FROM\_UNIXTIME(1320981071)

->2011-11-11 11:11:11

UTC\_DATE()

返回UTC日期

SELECT UTC\_DATE()

->2014-12-17

UTC\_TIME()

返回UTC时间

SELECT UTC\_TIME()

->08:01:45 (慢了8小时)

MONTH(d)

返回日期d中的月份值，1->12

SELECT MONTH('2011-11-11 11:11:11')

->11

MONTHNAME(d)

返回日期当中的月份名称，如Janyary

SELECT MONTHNAME('2011-11-11 11:11:11')

->November

DAYNAME(d)

返回日期d是星期几，如Monday,Tuesday

SELECT DAYNAME('2011-11-11 11:11:11')

->Friday

DAYOFWEEK(d)

日期d今天是星期几，1星期日，2星期一

SELECT DAYOFWEEK('2011-11-11 11:11:11')

->6

WEEKDAY(d)

日期d今天是星期几，0表示星期一，1表示星期二

WEEK(d)，WEEKOFYEAR(d)

计算日期d是本年的第几个星期，范围是0->53

SELECT WEEK('2011-11-11 11:11:11')

->45

DAYOFYEAR(d)

计算日期d是本年的第几天

SELECT DAYOFYEAR('2011-11-11 11:11:11')

->315

DAYOFMONTH(d)

计算日期d是本月的第几天

SELECT DAYOFMONTH('2011-11-11 11:11:11')

->11

QUARTER(d)

返回日期d是第几季节，返回1->4

SELECT QUARTER('2011-11-11 11:11:11')

->4

HOUR(t)

返回t中的小时值

SELECT HOUR('1:2:3')

->1

MINUTE(t)

返回t中的分钟值

SELECT MINUTE('1:2:3')

->2

SECOND(t)

返回t中的秒钟值

SELECT SECOND('1:2:3')

->3

EXTRACT(type FROM d)

从日期d中获取指定的值，type指定返回的值

SELECT EXTRACT(MINUTE FROM '2011-11-11 11:11:11')

->11

**4 type可取值为**

MICROSECOND

SECOND

MINUTE

HOUR

DAY

WEEK

MONTH

QUARTER

YEAR

SECOND\_MICROSECOND

MINUTE\_MICROSECOND

MINUTE\_SECOND

HOUR\_MICROSECOND

HOUR\_SECOND

HOUR\_MINUTE

DAY\_MICROSECOND

DAY\_SECOND

DAY\_MINUTE

DAY\_HOUR

YEAR\_MONTH

TIME\_TO\_SEC(t)

**5 将时间t转换为秒**

SELECT TIME\_TO\_SEC('1:12:00')

->4320

SEC\_TO\_TIME(s)

将以秒为单位的时间s转换为时分秒的格式

SELECT SEC\_TO\_TIME(4320)

->01:12:00

TO\_DAYS(d)

计算日期d距离0000年1月1日的天数

SELECT TO\_DAYS('0001-01-01 01:01:01')

->366

FROM\_DAYS(n)

计算从0000年1月1日开始n天后的日期

SELECT FROM\_DAYS(1111)

->0003-01-16

DATEDIFF(d1,d2)

计算日期d1->d2之间相隔的天数

SELECT DATEDIFF('2001-01-01','2001-02-02')

->-32

ADDDATE(d,n)

计算其实日期d加上n天的日期

ADDDATE(d，INTERVAL expr type)

计算起始日期d加上一个时间段后的日期

SELECT ADDDATE('2011-11-11 11:11:11',1)

->2011-11-12 11:11:11 (默认是天)

SELECT ADDDATE('2011-11-11 11:11:11', INTERVAL 5 MINUTE)

->2011-11-11 11:16:11 (TYPE的取值与上面那个列出来的函数类似)

DATE\_ADD(d,INTERVAL expr type) 同上

SUBDATE(d,n)

日期d减去n天后的日期

SELECT SUBDATE('2011-11-11 11:11:11', 1)

->2011-11-10 11:11:11 (默认是天)

SUBDATE(d,INTERVAL expr type)

日期d减去一个时间段后的日期

SELECT SUBDATE('2011-11-11 11:11:11', INTERVAL 5 MINUTE)

->2011-11-11 11:06:11 (TYPE的取值与上面那个列出来的函数类似)

ADDTIME(t,n)

时间t加上n秒的时间

SELECT ADDTIME('2011-11-11 11:11:11', 5)

->2011-11-11 11:11:16 (秒)

SUBTIME(t,n)

时间t减去n秒的时间

SELECT SUBTIME('2011-11-11 11:11:11', 5)

->2011-11-11 11:11:06 (秒)

DATE\_FORMAT(d,f)

按表达式f的要求显示日期d

SELECT DATE\_FORMAT('2011-11-11 11:11:11','%Y-%m-%d %r')

->2011-11-11 11:11:11 AM

TIME\_FORMAT(t,f)

按表达式f的要求显示时间t

SELECT TIME\_FORMAT('11:11:11','%r')

11:11:11 AM

GET\_FORMAT(type,s)

获得国家地区时间格式函数

select get\_format(date,'usa')

->%m.%d.%Y (注意返回的就是这个奇怪的字符串(format字符串))

**6 条件判断函数**

IF(expr,v1,v2)函数

如果表达式expr成立，返回结果v1；否则，返回结果v2。

SELECT IF(1 > 0,'正确','错误')

->正确

IFNULL(v1,v2)函数

如果v1的值不为NULL，则返回v1，否则返回v2。

SELECT IFNULL(null,'Hello Word')

->Hello Word

CASE

　　语法1：

CASE

　　WHEN e1

　　THEN v1

　　WHEN e2

　　THEN e2

　　...

　　ELSE vn

END

　　CASE表示函数开始，END表示函数结束。如果e1成立，则返回v1,如果e2成立，则返回v2，当全部不成立则返回vn，而当有一个成立之后，后面的就不执行了。

SELECT CASE

　　WHEN 1 > 0

　　THEN '1 > 0'

　　WHEN 2 > 0

　　THEN '2 > 0'

　　ELSE '3 > 0'

　　END

->1 > 0

　　语法2：

CASE expr

　　WHEN e1 THEN v1

　　WHEN e1 THEN v1

　　...

　　ELSE vn

END

　　如果表达式expr的值等于e1，返回v1；如果等于e2,则返回e2。否则返回vn。

SELECT CASE 1

　　WHEN 1 THEN '我是1'

　　WHEN 2 THEN '我是2'

ELSE '你是谁'

**7 系统信息函数-系统信息函数用来查询MySQL数据库的系统信息。**

VERSION()

返回数据库的版本号

SELECT VERSION()

->5.0.67-community-nt

CONNECTION\_ID() 返回服务器的连接数

DATABASE()、SCHEMA 返回当前数据库名

USER()、SYSTEM\_USER()、SESSION\_USER()、

CURRENT\_USER()、CURRENT\_USER返回当前用户

CHARSET(str) 返回字符串str的字符集

COLLATION(str) 返回字符串str的字符排列方式

LAST\_INSERT\_ID() 返回最近生成的AUTO\_INCREMENT值

7 加密函数-加密函数是MySQL用来对数据进行加密的函数。

PASSWORD(str)

该函数可以对字符串str进行加密，一般情况下，PASSWORD(str)用于给用户的密码加密。

SELECT PASSWORD('123')

->\*23AE809DDACAF96AF0FD78ED04B6A265E05AA257

MD5

MD5(str)函数可以对字符串str进行散列，可以用于一些普通的不需要解密的数据加密。

SELECT md5('123')

->202cb962ac59075b964b07152d234b70

ENCODE(str,pswd\_str)与DECODE(crypt\_str,pswd\_str)

ENCODE函数可以使用加密密码pswd\_str来加密字符串str，加密结果是二进制数，需要使用BLOB类型的字段保存。该函数与DECODE是一对，需要同样的密码才能够解密。

SELECT ENCODE('123','xxoo')

->;vx

SELECT DECODE(';vx','xxoo')

->123

**8 其他函数**

格式化函数FORMAT(x,n)

FORMAT(x,n)函数可以将数字x进行格式化，将x保留到小数点后n位。

SELECT FORMAT(3.1415926,3)

->3.142

不同进制的数字进行转换

ASCII(s) 返回字符串s的第一个字符的ASCII码；

BIN（x） 返回x的二进制编码；

HEX(x) 返回x的十六进制编码；

OCT(x) 返回x的八进制编码；

CONV(x,f1,f2) 返回f1进制数变成f2进制数；

IP地址与数字相互转换的函数

INET\_ATON(IP)函数可以将IP地址转换为数字表示；IP值需要加上引号；

INET\_NTOA(n)函数可以将数字n转换成IP形式。

SELECT INET\_ATON('192.168.0.1')

->3232235521

SELECT INET\_NTOA(3232235521)

->192.168.0.1

加锁函数和解锁函数

GET\_LOCK(name,time)函数定义一个名称为nam、持续时间长度为time秒的锁。如果锁定成功，则返回1；如果尝试超时，则返回0；如果遇到错误，返回NULL。

RELEASE\_LOCK(name)函数解除名称为name的锁。如果解锁成功，则返回1；如果尝试超时，返回0了如果解锁失败，返回NULL；

IS\_FREE\_LOCK(name)函数判断是否已使用名为name的锁定。如果使用，返回0，否则，返回1；

SELECT GET\_LOCK('MySQL',10)

->1 (持续10秒)

SELECT IS\_FREE\_LOCK('MySQL')

->1

SELECT RELEASE\_LOCK('MySQL')

->1

重复执行指定操作的函数

BENCHMARK(count.expr)函数将表达式expr重复执行count此，然后返回执行时间。该函数可以用来判断MySQL处理表达式的速度。

SELECT BENCHMARK(10000,NOW())

->0 返回系统时间1万

改变字符集的函数

CONVERT(s USING cs)函数将字符串s的字符集变成cs。

SELECT CHARSET('ABC')

->utf-8

SELECT CHARSET(CONVERT('ABC' USING gbk))

->gbk

**9 转换数据类型**

CAST(x AS type)

CONVERT(x,type)

这两个函数只对BINARY、CHAR、DATE、DATETIME、TIME、SIGNED INTEGER、UNSIGNED INTEGER。

SELECT CAST('123' AS UNSIGNED INTEGER) + 1

->124

SELECT '123' + 1

->124 其实MySQL能默认转换

SELECT CAST(NOW() AS DATE)

　　->2014-12-18

## 六、SQL调优

**1 MySQL查询截取分析步骤：**

（1）开启慢查询日志，捕获慢SQL

（2）explain+慢SQL分析

（3）show profile查询SQL语句在服务器中的执行细节和生命周期

（4）SQL数据库服务器参数调优

**2 开启慢查询日志，捕获慢SQL**

（1）查看慢查询日志是否开启

SHOW VARIABLES LIKE '%slow\_query\_log%';

（2）开启慢查询日志

SET GLOBAL slow\_query\_log=1;

（3）查看慢查询日志阙值

SHOW [GLOBAL] VARIABLES LIKE '%long\_query\_time%';

这个值表示超过多长时间的SQL语句会被记录到慢查询日志中

（4）设置慢查询日志阙值

SET GLOBAL long\_query\_time=3;

（5）查看多少SQL语句超过了阙值

SHOW GLOBAL STATUS LIKE '%Slow\_queries%';

（6）MySQL提供的日志分析工具mysqldumpslow

进入MySQL的安装目录中的bin目录下

执行 ./mysqldumpslow --help 查看帮助命令

常用参考：

得到返回记录集最多的10个SQL

mysqldumpslow -s r -t 10 slow.log

得到访问次数最多的10个SQL

mysqldumpslow -s c -t 10 slow.log

得到按照时间排序的前10条里面含有左连接的查询语句

mysqldumpslow -s t -t 10 -g "left join" slow.log

使用这些语句时结合| more使用

**3 explain+慢SQL分析**

使用EXPLAIN关键字可以模拟优化器执行SQL查询语句，从而知道MySQL是 如何处理你的SQL语句的。分析你的查询语句或是表结构的性能瓶颈。

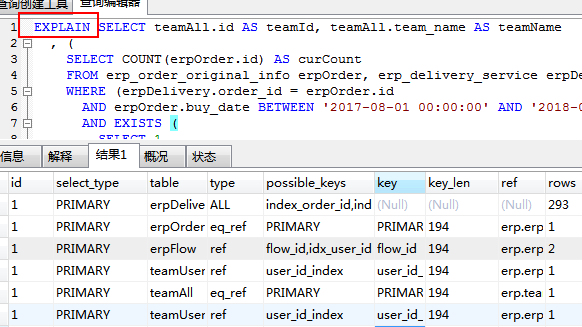
使用方式：Explain+SQL语句

执行计划包含的信息：

+----+-------------+-------+------+---------------+------+---------+------+------+-------+

| id | select\_type | table | type | possible\_keys | key | key\_len | ref | rows | Extra |

**+----+-------------+-------+------+---------------+------+---------+------+------+-------+**



**（1）id**

SELECT查询的序列号，包含一组数字，表示查询中执行SELECT语句或操作表的顺序

包含三种情况：

1.id相同，执行顺序由上至下

2.id不同，如果是子查询，id序号会递增，id值越大优先级越高，越先被执行

3.id既有相同的，又有不同的。id如果相同认为是一组，执行顺序由上至下； 在所有组中，id值越大优先级越高，越先执行。

**（2）select\_type**

SIMPLE:简单SELECT查询，查询中不包含子查询或者UNION

PRIMARY:查询中包含任何复杂的子部分，最外层的查询

SUBQUERY：SELECT或WHERE中包含的子查询部分

DERIVED：在FROM中包含的子查询被标记为DERIVER(衍生)， MySQL会递归执行这些子查询，把结果放到临时表中

UNION：若第二个SELECT出现UNION，则被标记为UNION, 若UNION包含在FROM子句的子查询中，外层子查询将被标记为DERIVED

UNION RESULT：从UNION表获取结果的SELECT

**（3）table**

显示这一行数据是关于哪张表的

**（4）type**

type显示的是访问类型，是较为重要的一个指标，结果值从最好到最坏依次是：

system>const>eq\_ref>ref>fulltext>ref\_or\_null>index\_merge>unique\_subquery>index\_subquery>range>index>ALL

一般来说，得保证查询至少达到range级别，最好能达到ref。

system：表只有一行记录（等于系统表），这是const类型的特例，平时不会出现

const：如果通过索引依次就找到了，const用于比较主键索引或者unique索引。 因为只能匹配一行数据，所以很快。如果将主键置于where列表中，MySQL就能将该查询转换为一个常量

eq\_ref：唯一性索引扫描，对于每个索引键，表中只有一条记录与之匹配。常见于主键或唯一索引扫描

ref：非唯一性索引扫描，返回匹配某个单独值的所有行。本质上也是一种索引访问，它返回所有匹配 某个单独值的行，然而它可能会找到多个符合条件的行，所以它应该属于查找和扫描的混合体

range：只检索给定范围的行，使用一个索引来选择行。key列显示使用了哪个索引，一般就是在你的where语句中出现between、<、>、in等的查询，这种范围扫描索引比全表扫描要好，因为只需要开始于缩印的某一点，而结束于另一点，不用扫描全部索引

index：Full Index Scan ，index与ALL的区别为index类型只遍历索引树，这通常比ALL快，因为索引文件通常比数据文件小。 （也就是说虽然ALL和index都是读全表， 但index是从索引中读取的，而ALL是从硬盘读取的）

all：Full Table Scan，遍历全表获得匹配的行

**（5）possible\_keys**

显示可能应用在这张表中的索引，一个或多个。 查询涉及到的字段上若存在索引，则该索引将被列出，但不一定被查询实际使用

**（6）key**

实际使用的索引。如果为NULL，则没有使用索引。

查询中若出现了覆盖索引，则该索引仅出现在key列表中。

**（7）key\_len**

表示索引中使用的字节数，可通过该列计算查询中使用的索引的长度。在不损失精度的情况下，长度越短越好。

key\_len显示的值为索引字段的最大可能长度，并非实际使用长度，即key\_len是根据表定义计算而得，不是通过表内检索出的。

**（8）ref**

显示索引的哪一列被使用了，哪些列或常量被用于查找索引列上的值。

**（9）rows**

根据表统计信息及索引选用情况，大致估算出找到所需记录多需要读取的行数。

**（10）Extra**

包含不适合在其他列中显示但十分重要的额外信息：

@Using filesort： 说明MySQL会对数据使用一个外部的索引排序，而不是按照表内的索引顺序进行读取。MySQL中无法利用索引完成的排序操作称为“文件排序”

@Using temporary： 使用了临时表保存中间结果，MySQL在对查询结果排序时使用临时表。常见于排序order by和分组查询group by

@Using index： 表示相应的SELECT操作中使用了覆盖索引（Covering Index），避免访问了表的数据行，效率不错。 如果同时出现using where，表明索引被用来执行索引键值的查找； 如果没有同时出现using where，表明索引用来读取数据而非执行查找动作 覆盖索引（Covering Index）： 理解方式1：SELECT的数据列只需要从索引中就能读取到，不需要读取数据行，MySQL可以利用索引返回SELECT列表中 的字段，而不必根据索引再次读取数据文件，换句话说查询列要被所建的索引覆盖 理解方式2：索引是高效找到行的一个方法，但是一般数据库也能使用索引找到一个列的数据，因此他不必读取整个行。 毕竟索引叶子节点存储了他们索引的数据；当能通过读取索引就可以得到想要的数据，那就不需要读取行了，一个索引 包含了（覆盖）满足查询结果的数据就叫做覆盖索引 注意： 如果要使用覆盖索引，一定要注意SELECT列表中只取出需要的列，不可SELECT \*, 因为如果所有字段一起做索引会导致索引文件过大查询性能下降

@impossible where： WHERE子句的值总是false，不能用来获取任何元组

@select tables optimized away： 在没有GROUP BY子句的情况下基于索引优化MIN/MAX操作或者对于MyISAM存储引擎优化COUNT(\*)操作， 不必等到执行阶段再进行计算，查询执行计划生成的阶段即完成优化

@distinct： 优化distinct操作，在找到第一匹配的元祖后即停止找同样值的操作

**4 show profile查询SQL语句在服务器中的执行细节和生命周期**

Show Profile是MySQL提供可以用来分析当前会话中语句执行的资源消耗情况，可以用于SQL的调优测量

默认关闭，并保存最近15次的运行结果

分析步骤

1）查看状态：SHOW VARIABLES LIKE 'profiling';

2）启：set profiling=on;

3）查看结果：show profiles;

4）诊断SQL：show profile cpu,block io for query 上一步SQL数字号码;

ALL：显示所有开销信息

BLOCK IO：显示IO相关开销

CONTEXT SWITCHES：显示上下文切换相关开销

CPU：显示CPU相关开销

IPC：显示发送接收相关开销

MEMORY：显示内存相关开销

PAGE FAULTS：显示页面错误相关开销

SOURCE：显示和Source\_function，Source\_file，Source\_line相关开销

SWAPS：显示交换次数相关开销

5）注意（遇到这几种情况要优化）

converting HEAP to MyISAM： 查询结果太大，内存不够用往磁盘上搬

Creating tmp table：创建临时表

Copying to tmp table on disk：把内存中的临时表复制到磁盘

locked

**5 SQL数据库服务器参数调优**

当order by 和 group by无法使用索引时，增大max\_length\_for\_sort\_data参数设置和增大sort\_buffer\_size参数的设置

**6 查看MySQL运行情况**

（1）查看MySQL运行情况

SHOW STATUS;

（2）查看INNODB数据库引擎运行状态

SHOW ENGINE INNODB STATUS;

（3）查看当前正在进行的进程，对于有锁表等情况的排查很有用处

SHOW PROCESSLIST; 默认显示前100条

SHOW FULL PROCESSLIST; 显示所有

（4）查看MySQL的配置参数

SHOW VARIABLES;

（5）查看当前已经被打开的表列表

SHOW OPEN TABLES;

注意：一般情况下，打开MySQL的慢查询记录同样有利于排查，如何开启慢查询，请点击。

附1 SHOW STATUS;关键结果释义

Aborted\_clients 由于客户没有正确关闭连接已经死掉，已经放弃的连接数量。

Aborted\_connects 尝试已经失败的MySQL服务器的连接的次数。

Connections 试图连接MySQL服务器的次数。

Created\_tmp\_tables 当执行语句时，已经被创造了的隐含临时表的数量。

Delayed\_insert\_threads 正在使用的延迟插入处理器线程的数量。

Delayed\_writes 用INSERT DELAYED写入的行数。

Delayed\_errors 用INSERT DELAYED写入的发生某些错误(可能重复键值)的行数。

Flush\_commands 执行FLUSH命令的次数。

Handler\_delete 请求从一张表中删除行的次数。

Handler\_read\_first 请求读入表中第一行的次数。

Handler\_read\_key 请求数字基于键读行。

Handler\_read\_next 请求读入基于一个键的一行的次数。

Handler\_read\_rnd 请求读入基于一个固定位置的一行的次数。

Handler\_update 请求更新表中一行的次数。

Handler\_write 请求向表中插入一行的次数。

Key\_blocks\_used 用于关键字缓存的块的数量。

Key\_read\_requests 请求从缓存读入一个键值的次数。

Key\_reads 从磁盘物理读入一个键值的次数。

Key\_write\_requests 请求将一个关键字块写入缓存次数。

Key\_writes 将一个键值块物理写入磁盘的次数。

Max\_used\_connections 同时使用的连接的最大数目。

Not\_flushed\_key\_blocks 在键缓存中已经改变但是还没被清空到磁盘上的键块。

Not\_flushed\_delayed\_rows 在INSERT DELAY队列中等待写入的行的数量。

Open\_tables 打开表的数量。

Open\_files 打开文件的数量。

Open\_streams 打开流的数量(主要用于日志记载）

Opened\_tables 已经打开的表的数量。

Questions 发往服务器的查询的数量。

Slow\_queries 要花超过long\_query\_time时间的查询数量。

Threads\_connected 当前打开的连接的数量。

Threads\_running 不在睡眠的线程数量。

Uptime 服务器工作了多少秒。

**7 提升性能的建议**

1.如果opened\_tables太大,应该把my.cnf中的table\_cache变大

2.如果Key\_reads太大,则应该把my.cnf中key\_buffer\_size变大.可以用Key\_reads/Key\_read\_requests计算出cache失败率

3.如果Handler\_read\_rnd太大,则你写的SQL语句里很多查询都是要扫描整个表,而没有发挥索引的键的作用

4.如果Threads\_created太大,就要增加my.cnf中thread\_cache\_size的值.可以用Threads\_created/Connections计算cache命中率

5.如果Created\_tmp\_disk\_tables太大,就要增加my.cnf中tmp\_table\_size的值,用基于内存的临时表代替基于磁盘的

**8 SHOW PROCESSLIST;结果释义**

id列:一个标识，你要kill 一个语句的时候很有用。

user列: 显示当前用户，如果不是root，这个命令就只显示你权限范围内的sql语句。

host列:显示这个语句是从哪个ip 的哪个端口上发出的。可用来追踪出问题语句的用户。

db列:显示这个进程目前连接的是哪个数据库。

command列:显示当前连接的执行的命令，一般就是休眠（sleep），查询（query），连接（connect）。

time列:此这个状态持续的时间，单位是秒。

state列:显示使用当前连接的sql语句的状态，很重要的列，后续会有所有的状态的描述，请注意，state只是语句执行中的某一个状态，一个sql语句，已查询为例，可能需要经过copying to tmp table，Sorting result，Sending data等状态才可以完成。

info列:显示这个sql语句，因为长度有限，所以长的sql语句就显示不全，但是一个判断问题语句的重要依据。

**9 state列具体释义：**

Checking table 正在检查数据表（这是自动的）。

Closing tables 正在将表中修改的数据刷新到磁盘中，同时正在关闭已经用完的表。这是一个很快的操作，如果不是这样的话，就应该确认磁盘空间是否已经满了或者磁盘是否正处于重负中。

Connect Out 复制从服务器正在连接主服务器。

Copying to tmp table on disk 由于临时结果集大于 tmp\_table\_size，正在将临时表从内存存储转为磁盘存储以此节省内存。

Creating tmp table 正在创建临时表以存放部分查询结果。

deleting from main table 服务器正在执行多表删除中的第一部分，刚删除第一个表。

deleting from reference tables 服务器正在执行多表删除中的第二部分，正在删除其他表的记录。

Flushing tables 正在执行 FLUSH TABLES，等待其他线程关闭数据表。

Killed发送了一个kill请求给某线程，那么这个线程将会检查kill标志位，同时会放弃下一个kill请求。MySQL会在每次的主循环中检查kill标志位，不过有些情况下该线程可能会过一小段才能死掉。

Locked 被其他查询锁住了。

Sending data 正在处理 SELECT 查询的记录，同时正在把结果发送给客户端。

Sorting for group 正在为 GROUP BY 做排序。

Sorting for order 正在为 ORDER BY 做排序。

Opening tables 这个过程应该会很快，除非受到其他因素的干扰。例如，在执 ALTER TABLE 或 LOCK TABLE 语句行完以前，数据表无法被其他线程打开。 正尝试打开一个表。

Removing duplicates 正在执行一个 SELECT DISTINCT 方式的查询，但是MySQL无法在前一个阶段优化掉那些重复的记录。因此，MySQL需要再次去掉重复的记录，然后再把结果发送给客户端。

Reopen table 获得了对一个表的锁，但是必须在表结构修改之后才能获得这个锁。已经释放锁，关闭数据表，正尝试重新打开数据表。

Repair by sorting 修复指令正在排序以创建索引。

Repair with keycache 修复指令正在利用索引缓存一个一个地创建新索引。它会比 Repair by sorting 慢些。

Searching rows for update 正在讲符合条件的记录找出来以备更新。它必须在 UPDATE 要修改相关的记录之前就完成了。

Sleeping 正在等待客户端发送新请求.

System lock 正在等待取得一个外部的系统锁。如果当前没有运行多个 mysqld 服务器同时请求同一个表，那么可以通过增加 –skip-external-locking参数来禁止外部系统锁。

Upgrading lock INSERT DELAYED 正在尝试取得一个锁表以插入新记录。

Updating 正在搜索匹配的记录，并且修改它们。

User Lock 正在等待 GET\_LOCK()。

Waiting for tables 该线程得到通知，数据表结构已经被修改了，需要重新打开数据表以取得新的结构。然后，为了能的重新打开数据表，必须等到所有其他线程关闭这个表。

**10 以下几种情况下会产生这个通知：FLUSH TABLES tbl\_name, ALTERTABLE, RENAME TABLE, REPAIR TABLE, ANALYZE TABLE, 或 OPTIMIZE TABLE。**

waiting for handler insert INSERT DELAYED 已经处理完了所有待处理的插入操作，正在等待新的请求。 大部分状态对应很快的操作，只要有一个线程保持同一个状态好几秒钟，那么可能是有问题发生了，需要检查一下。还有其它的状态没在上面中列出来，不过它们大部分只是在查看服务器是否有存在错误是才用得着。

## 八、读写分离

**1 大型网站为了软解大量的并发访问**，除了在网站实现分布式负载均衡，远远不够。到了数据业务层、数据访问层，如果还是传统的数据结构，或者只是单单靠一台服务器扛，如此多的数据库连接操作，数据库必然会崩溃，数据丢失的话，后果更是不堪设想。这时候，我们会考虑如何减少数据库的联接，一方面采用优秀的代码框架，进行代码的优化，采用优秀的数据缓存技术如：memcached,如果资金丰厚的话，必然会想到假设服务器群，来分担主数据库的压力。Ok切入今天微博主题，利用MySQL主从配置，实现读写分离，减轻数据库压力。

**2 概述：**搭设一台Master服务器（win8.1系统，Ip：192.168.0.104），搭设两台Slave服务器（虚拟机——一台Ubuntu，一台 Windows Server 2003）

**3 原理：**主服务器（Master）负责网站NonQuery操作，从服务器负责Query操作，用户可以根据网站功能模特性块固定访问Slave服务器，或者自己写个池或队列，自由为请求分配从服务器连接。主从服务器利用MySQL的二进制日志文件，实现数据同步。二进制日志由主服务器产生，从服务器响应获取同步数据库。

**4 在主从服务器上都装上MySQL数据库**，windows系统鄙人安装的是mysql\_5.5.25.msi版本，Ubuntu安装的是mysql-5.6.22-linux-glibc2.5-i686.tar 。windows安装mysql就不谈了，一般地球人都应该会。鄙人稍微说一下Ubuntu的MySQL安装，我建议不要在线下载安装，还是离线安装的好。大家可以参考 http://www.linuxidc.com/Linux/2013-01/78716.htm 这位不知道大哥还是姐妹，写的挺好按照这个就能装上。在安装的时候可能会出现几种现象，大家可以参考解决一下：

（1）如果您不是使用root用户登录，建议 su - root 切换到Root用户安装，那就不用老是 sudo 了。

（2）存放解压的mysql 文件夹，文件夹名字最好改成mysql

（3）在./support-files/mysql.server start 启动MySQL的时候，可能会出现一个警告，中文意思是启动服务运行读文件时，忽略了my.cnf文件，那是因为my.cnf的文件权限有问题，mysql会认为该文件有危险不会执行。但是mysql还会启动成功，但如果下面配置从服务器参数修改my.cnf文件的时候，你会发现文件改过了，但是重启服务时，修改过后的配置没有执行，而且您 list一下mysql的文件夹下会发现很多.my.cnf.swp等中间文件。这都是因为MySQL启动时没有读取my.cnf的原因。这时只要将my.cnf的文件权限改成my\_new.cnf的权限一样就Ok，命令：chmod 644 my.cnf就Ok

（4）Ubuntu中修改文档内容没有Vim，最好把Vim 装上，apt-get install vim,不然估计会抓狂。

这时候我相信MySQL应该安装上去了。

**5 配置Master主服务器**

（1）在Master MySQL上创建一个用户‘repl’，并允许其他Slave服务器可以通过远程访问Master，通过该用户读取二进制日志，实现数据同步。创建用于读取日志的数据库用户

（2）找到MySQL安装文件夹修改my.Ini文件。mysql中有好几种日志方式，这不是今天的重点。我们只要启动二进制日志log-bin就ok。

在[mysqld]下面增加下面几行代码

（3）查看日志

mysql> SHOW MASTER STATUS;

+-------------------+----------+--------------+------------------+

| File | Position | Binlog\_Do\_DB | Binlog\_Ignore\_DB |

+-------------------+----------+--------------+------------------+

| master-bin.000001 | 1285 | | |

+-------------------+----------+--------------+------------------+

1 row in set (0.00 sec)

重启MySQL服务

**6 配置Slave从服务器（windows）**

（1）找到MySQL安装文件夹修改my.ini文件，在[mysqld]下面增加下面几行代码

my.cnf 配置，重启MySQL服务

（2）连接Master

change master to master\_host='192.168.0.104', //Master 服务器Ip

master\_port=3306,

master\_user='repl',

master\_password='mysql',

master\_log\_file='master-bin.000001',//Master服务器产生的日志

master\_log\_pos=0;

（3）启动Slave

start slave;

**7 Slave从服务器（Ubuntu）**

（1）找到MySQL安装文件夹修改my.cnf文件,vim my.cnf

（2） ./support-files/myql.server restart 重启MySQL服务 , ./bin/mysql 进入MySQL命令窗口

（3）连接Master

change master to master\_host='192.168.0.104', //Master 服务器Ip

master\_port=3306,

master\_user='repl',

master\_password='mysql',

master\_log\_file='master-bin.000001',//Master服务器产生的日志

master\_log\_pos=0;

（4）启动Slave

start slave;

OK所有配置都完成了，这时候大家可以在Master Mysql 中进行测试了，因为我们监视的时Master mysql 所有操作日志，所以，你的任何改变主服务器数据库的操作，都会同步到从服务器上。创建个数据库，表试试吧。。

## 九、备份恢复与数据导出

**1 Mysql备份策略：**

（1）完整备份：

完整备份就是指对某一个时间点上的所有数据或应用进行的一个完整拷贝，对数据量大的，备份时间较长，当然数据在恢复的时候快。

（2）增量备份：

备份自上一次备份（包括完整备份，差异备份，增量备份）之后所有变化的数据进行备份。恢复的时候只需要一次完整的备份加上完整备份后的多个增量备份进行恢复即可。

（3）差异备份：

备份自上一次完整备份之后所有变化的数据，恢复的时候仅需要最新一次完整备份加上差异备份即可。

**2 备份方式**

（1）使用 mysqldump 进行逻辑备份

（2）使用 LVM 快照备份：快照备份就是把当时的场景保存为一个不变的状态，然后对这个不变的状态进行备份。但然，在规划mysql数据库时最好将数据和日志分开放到lvm分区中。使用LVM快照备份，需要将数据放在lvm分区。

（3）Xtrabackup备份：

**3 备份原理：**

XtraBackup 基于 InnoDB 的 crash-recovery 功能。 它会复制 innodb 的data file ，由于不锁表，复制出来的数据是不一致的， 在恢复的时候使用 crash-recovery ，使得数据恢复一致。 InnoDB 维护了一个 redo log ，又称为 transaction log ，事务日志， 它包含了 innodb 数据的所有改动情况。当InnoDB启动的时候，它会先去检查datafile和transaction log，并且会做二步操作：

1.Itapplies committed transaction log entries to the data files

2.itperforms an undo operation on any transactions that modified data but did notcommit.

XtraBackup在备份的时候，一页一页地复制innodb的数据，而且不锁定表，

与此同时，XtraBackup还有另外一个线程监视着transactionslog，

一旦log发生变化，就把变化过的logpages复制走。

为什么要急着复制走呢？前几章的时候就提过这个问题，

因为transactions log文件大小有限，写满之后，就会从头再开始写，

所以新数据可能会覆盖到旧的数据。

在prepare过程中，XtraBackup使用复制到的transactionslog 对备份出来的innodb data file 进行crash recovery。

用mysqldump实现备份：

首先，mysqldump是mysql的一个客户端工具，可以实现备份整个服务器、单个或部分数据库、单个或部分表、表中的某些行，存储过程，存储函数，触发器等，能自动记录备份时的二进制日志文件及相应的position值

**4 详解用mysqldump实现备份恢复操作**

首先，mysqldump是mysql的一个客户端工具，可以实现备份整个服务器、单个或部分数据库、单个或部分表、表中的某些行，存储过程，存储函数，触发器等，能自动记录备份时的二进制日志文件及相应的position值

**5 对mysqldump参数说明**

--all-databases , -A // 导出全部数据库。

--all-tablespaces , -Y // 导出全部表空间。

--no-tablespaces , -y // 不导出任何表空间信息。

--add-drop-database // 每个数据库创建之前添加drop数据库语句。

--add-drop-table // 每个数据表创建之前添加drop数据表语句。(默认为打开状态，使用--skip-add-drop-table取消选项)

--add-locks // 在每个表导出之前增加LOCK TABLES并且之后UNLOCK TABLE。(默认为打开状态，使用--skip-add-locks取消选项)

--allow-keywords // 允许创建是关键词的列名字。这由表名前缀于每个列名做到。

--apply-slave-statements // 在'CHANGE MASTER'前添加'STOP SLAVE'，并且在导出的最后添加'START SLAVE'。

--character-sets-dir // 字符集文件的目录

--comments // 附加注释信息。默认为打开，可以用--skip-comments取消

--compatible // 导出的数据将和其它数据库或旧版本的MySQL 相兼容。值可以为ansi、mysql323、mysql40、postgresql、oracle、mssql、db2、maxdb、no\_key\_options、no\_tables\_options、no\_field\_options等，

**6 要使用几个值，用逗号将它们隔开。它并不保证能完全兼容，而是尽量兼容。**

--compact // 导出更少的输出信息(用于调试)。去掉注释和头尾等结构。可以使用选项：--skip-add-drop-table --skip-add-locks --skip-comments --skip-disable-keys

--complete-insert, -c // 使用完整的insert语句(包含列名称)。这么做能提高插入效率，但是可能会受到max\_allowed\_packet参数的影响而导致插入失败。

--compress, -C // 在客户端和服务器之间启用压缩传递所有信息

--create-options, -a // 在CREATE TABLE语句中包括所有MySQL特性选项。(默认为打开状态)

--databases, -B // 导出几个数据库。参数后面所有名字参量都被看作数据库名。

--debug // 输出debug信息，用于调试。默认值为：d:t:o,/tmp/mysqldump.trace

--debug-check // 检查内存和打开文件使用说明并退出。

--debug-info // 输出调试信息并退出

--delete-master-logs //master 备份后删除日志. 这个参数将自动激活--master-data。

--disable-keys // 对于每个表，用/\*!40000 ALTER TABLE tbl\_name DISABLE KEYS \*/;和/\*!40000 ALTER TABLE tbl\_name ENABLE KEYS \*/;语句引用INSERT语句。这样可以更快地导入dump出来的文件，因为它是在插入所有行后创建索引的。该选项只适合MyISAM表，默认为打开状态。

--dump-slave // 该选项将导致主的binlog位置和文件名追加到导出数据的文件中。设置为1时，将会以CHANGE MASTER命令输出到数据文件；设置为2时，在命令前增加说明信息。该选项将会打开--lock-all-tables，除非--single-transaction被指定。该选项会自动关闭--lock-tables选项。默认值为0。

--events, -E // 导出事件。

--flush-logs // 开始导出之前刷新日志。

请注意：假如一次导出多个数据库(使用选项--databases或者--all-databases)，将会逐个数据库刷新日志。除使用--lock-all-tables或者--master-data外。在这种情况下，日志将会被刷新一次，相应的所以表同时被锁定。因此，如果打算同时导出和刷新日志应该使用--lock-all-tables 或者--master-data 和--flush-logs。

--flush-privileges // 在导出mysql数据库之后，发出一条FLUSH PRIVILEGES 语句。为了正确恢复，该选项应该用于导出mysql数据库和依赖mysql数据库数据的任何时候。

--ignore-table // 不导出指定表。指定忽略多个表时，需要重复多次，每次一个表。每个表必须同时指定数据库和表名。

--lock-all-tables, -x // 提交请求锁定所有数据库中的所有表，以保证数据的一致性。这是一个全局读锁，并且自动关闭--single-transaction 和--lock-tables 选项。

mysqldump -uroot -p --host=localhost --all-databases --lock-all-tables

--lock-tables, -l // 开始导出前，锁定所有表。用READ LOCAL锁定表以允许MyISAM表并行插入。对于支持事务的表例如InnoDB和BDB，--single-transaction是一个更好的选择，因为它根本不需要锁定表。

请注意当导出多个数据库时，--lock-tables分别为每个数据库锁定表。因此，该选项不能保证导出文件中的表在数据库之间的逻辑一致性。不同数据库表的导出状态可以完全不同。

--log-error // 附加警告和错误信息到给定文件

--master-data // 该选项将binlog的位置和文件名追加到输出文件中。如果为1，将会输出CHANGE MASTER 命令；如果为2，输出的CHANGE MASTER命令前添加注释信息。该选项将打开--lock-all-tables 选项，除非--single-transaction也被指定（在这种情况下，全局读锁在开始导出时获得很短的时间；其他内容参考下面的--single-transaction选项）。该选项自动关闭--lock-tables选项。

--no-create-db, -n // 只导出数据，而不添加CREATE DATABASE 语句。

--no-create-info, -t

只导出数据，而不添加CREATE TABLE 语句。

mysqldump -uroot -p --host=localhost --all-databases --no-create-info

--no-data, -d // 不导出任何数据，只导出数据库表结构。

--order-by-primary // 如果存在主键，或者第一个唯一键，对每个表的记录进行排序。在导出MyISAM表到InnoDB表时有效，但会使得导出工作花费很长时间。

--quick, -q // 不缓冲查询，直接导出到标准输出。默认为打开状态，使用--skip-quick取消该选项。

--opt // 同时启用各种高级选项

--routines, -R // 导出存储过程以及自定义函数。

--single-transaction // 该选项在导出数据之前提交一个BEGIN SQL语句，BEGIN 不会阻塞任何应用程序且能保证导出时数据库的一致性状态。它只适用于多版本存储引擎，仅InnoDB。本选项和--lock-tables 选项是互斥的，因为LOCK TABLES 会使任何挂起的事务隐含提交。要想导出大表的话，应结合使用--quick 选项。

--dump-date // 将导出时间添加到输出文件中。默认为打开状态，使用--skip-dump-date关闭选项。

--skip-opt // 禁用–opt选项.

--socket,-S // 指定连接mysql的socket文件位置，默认路径/tmp/mysql.sock

--triggers // 导出触发器。该选项默认启用，用--skip-triggers禁用它。

--user, -u // 指定连接的用户名。

--verbose, --v // 输出多种平台信息。

--version, -V // 输出mysqldump版本信息并退出

**7 实例详解 ： 操作时数据库以及库中的表可以自己创建，主要讲解备份的过程及其中的要点**

**（1）对catidb数据库进行完全备份**：

mysql>use catidb;

Databasechanged

mysql>show tables; //使用catidb数据库查看其中的表

+------------------+

|Tables\_in\_catidb |

+------------------+

|tb1 |

|tb2 |

+------------------+

2rows in set (0.00 sec)

**（2）对catidb数据库进行备份，恢复**

[root@localhost~]# mysqldump -uroot -pmypass catidb > /tmp/catidb.sql //用这种方法对catidb数据库进行备份，在数据库恢复时有一个问题，即该catidb.sql文件中没用创建catidb数据库的语句，在恢复时所含所有表会恢复到当前默认的数据库中，相当的危险，因此需要谨慎，在恢复之前一定要先创建数据库

mysql>drop database catidb; //备份之后将catidb数据库删除

QueryOK, 2 rows affected (0.07 sec)

mysql>create database catidb; //在恢复之前一定要先创建数据库，数据库名字可以自己定义

QueryOK, 1 row affected (0.04 sec)

mysql>use catidb;

Databasechanged

mysql>source /tmp/catidb.sql //进行数据恢复

mysql>show tables; //恢复成功

+------------------+

|Tables\_in\_catidb |

+------------------+

|tb1 |

|tb2 |

+------------------+

2rows in set (0.01 sec)

**（3）利用mysqldump实现从逻辑角度完全备份mysql，配合二进制日志备份实现增量备份**

**仍以上面的catidb数据库为例先对catidb 数据库进行完全备份**

mysql>use catidb;

Databasechanged

mysql>show tables;

+------------------+

|Tables\_in\_catidb |

+------------------+

|tb1 |

|tb2 |

+------------------+

2rows in set (0.00 sec)

**（4）对catidb数据库进行完全备份**

[root@localhost~]# mysqldump -uroot -pmypass --single-transaction --master-data=2 --databasescatidb > /backup/catidb\_`date +%F`.sql

备份完成之后又对catidb数据库进行了新的操作

mysql>show tables;

+------------------+

|Tables\_in\_catidb |

+------------------+

|tb1 |

|tb2 |

+------------------+

2rows in set (0.00 sec)

mysql>use catidb

Databasechanged

mysql>create table tb3 (id int);

QueryOK, 0 rows affected (0.11 sec)

mysql>insert into tb3 values (1),(6),(9);

QueryOK, 3 rows affected (0.09 sec)

Records:3 Duplicates: 0 Warnings: 0

**（5）进行增量备份： 从上一次完全备份完成后，到所有新的操作之前，那么如何查看记录的位置**

开始位置：

[root@localhost~]# less /backup/catidb\_2013-09-07.sql

--MySQL dump 10.13 Distrib 5.5.33, forLinux (x86\_64)

--

--Host: localhost Database: catidb

--------------------------------------------------------

--Server version 5.5.33-log

/\*!40101SET @OLD\_CHARACTER\_SET\_CLIENT=@@CHARACTER\_SET\_CLIENT \*/;

/\*!40101SET @OLD\_CHARACTER\_SET\_RESULTS=@@CHARACTER\_SET\_RESULTS \*/;

/\*!40101SET @OLD\_COLLATION\_CONNECTION=@@COLLATION\_CONNECTION \*/;

/\*!40101SET NAMES utf8 \*/;

/\*!40103SET @OLD\_TIME\_ZONE=@@TIME\_ZONE \*/;

/\*!40103SET TIME\_ZONE='+00:00' \*/;

/\*!40014SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0 \*/;

/\*!40014SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0 \*/;

/\*!40101SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='NO\_AUTO\_VALUE\_ON\_ZERO' \*/;

/\*!40111SET @OLD\_SQL\_NOTES=@@SQL\_NOTES, SQL\_NOTES=0 \*/;

--

--Position to start replication or point-in-time recovery from

--

-- CHANGE MASTER TO MASTER\_LOG\_FILE='mysql-bin.000004',MASTER\_LOG\_POS=1673;

结束位置：

mysql>show master status;

+------------------+----------+--------------+------------------+

|File | Position |Binlog\_Do\_DB | Binlog\_Ignore\_DB |

+------------------+----------+--------------+------------------+

| mysql-bin.000004 | 1959 | | |

+------------------+----------+--------------+------------------+

1row in set (0.00 sec)

**（6）备份恢复完成！**

注意：对于一般的备份，是针对整个服务器进行备份的因此所用的选项应该是—all-databases;

## 十、Mysql安全

**1 避免从互联网访问MySQL数据库，确保特定主机才拥有访问特权**

直接通过本地网络之外的计算机改变生产环境中的数据库是异常危险的。有时，管理员会打开主机对数据库的访问：

> GRANT ALL ON \*.\* TO 'root'@'%';

这其实是完全放开了对root的访问。所以，把重要的操作限制给特定主机非常重要：

> GRANT ALL ON \*.\* TO 'root'@'localhost';

> GRANT ALL ON \*.\* TO 'root'@'myip.athome'

> FLUSH PRIVILEGES

此时，你仍有完全的访问，但只有指定的IP(不管其是否静态)可以访问。

**2 定期备份数据库**

任何系统都有可能发生灾难。服务器、MySQL也会崩溃，也有可能遭受入侵，数据有可能被删除。只有为最糟糕的情况做好了充分的准备，才能够在事后快速地从灾难中恢复。企业最好把备份过程作为服务器的一项日常工作。

**3 前面说过，如果使用了远程访问，要确保只有定义的主机才可以访问服务器。**这一般是通过TCP wrappers、iptables或任何其它的防火墙软件或硬件实现的。

为限制打开网络socket，管理员应当在my.cnf或my.ini的[mysqld]部分增加下面的参数：

skip-networking

这些文件位于windows的C:\Program Files\MySQL\MySQL Server 5.1文件夹中，或在Linux中，my.cnf位于/etc/，或位于/etc/mysql/。这行命令在MySQL启动期间，禁用了网络连接的初始化。请注意，在这里仍可以建立与MySQL服务器的本地连接。

另一个可行的方案是，强迫MySQL仅监听本机，方法是在my.cnf的[mysqld]部分增加下面一行：

bind-address=127.0.0.1

如果企业的用户从自己的机器连接到服务器或安装到另一台机器上的web服务器，你可能不太愿意禁用网络访问。此时，不妨考虑下面的有限许可访问：

mysql> GRANT SELECT, INSERT ON mydb.\* TO 'someuser'@'somehost';

这里，你要把someuser换成用户名，把somehost换成相应的主机。

**4 设置root用户的口令并改变其登录名**

在linux中，root用户拥有对所有数据库的完全访问权。因而，在Linux的安装过程中，一定要设置root口令。当然，要改变默认的空口令，其方法如下：

Access MySQL控制台：$ mysql -u root -p

在MySQL控制台中执行：

> SET PASSWORD FOR 'root'@'localhost' = PASSWORD('new\_password');

在实际操作中，只需将上面一行的new\_password换成实际的口令即可。

在Linux控制台中更改root口令的另一种方法是使用mysqladmin工具：

$ mysqladmin -u root password new\_password

此时，也是将上面一行的new\_password换成实际的口令即可。

当然，这是需要使用强口令来避免强力攻击。

为了更有效地改进root用户的安全性，另一种好方法是为其改名。为此，你必须更新表用户中的mySQL数据库。在MySQL控制台中进行操作：

> USE mysql;

> UPDATE user SET user="another\_username" WHERE user="root";

> FLUSH PRIVILEGES;

然后，通过Linux访问MySQL控制台就要使用新用户名了：

$ mysql -u another\_username -p

**5 移除测试(test)数据库**

在默认安装的MySQL中，匿名用户可以访问test数据库。我们可以移除任何无用的数据库，以避免在不可预料的情况下访问了数据库。因而，在MySQL控制台中，执行：

> DROP DATABASE test;

**6 另一项改变是禁用”LOAD DATA LOCAL INFILE”命令，这有助于防止非授权用户访问本地文件。在PHP应用程序中发现有新的SQL注入漏洞时，这样做尤其重要。**

此外，在某些情况下，LOCAL INFILE命令可被用于访问操作系统上的其它文件(如/etc/passwd)，应使用下现的命令：

mysql> LOAD DATA LOCAL INFILE '/etc/passwd' INTO TABLE table1

更简单的方法是：

mysql> SELECT load\_file("/etc/passwd")

为禁用LOCAL INFILE命令，应当在MySQL配置文件的[mysqld]部分增加下面的参数：

set-variable=local-infile=0

**7 有些MySQL数据库的匿名用户的口令为空。因而，任何人都可以连接到这些数据库。可以用下面的命令进行检查：**

mysql> select \* from mysql.user where user="";

在安全的系统中，不会返回什么信息。另一种方法是：

mysql> SHOW GRANTS FOR ''@'localhost';

mysql> SHOW GRANTS FOR ''@'myhost';

如果grants存在，那么任何人都可以访问数据库，至少可以使用默认的数据库“test”。其检查方法如下：

shell> mysql -u blablabla

如果要移除账户，则执行命令：

mysql> DROP USER "";

从MySQL的5.0版开始支持DROP USER命令。如果你使用的老版本的MySQL，你可以像下面这样移除账户：

mysql> use mysql;

mysql> DELETE FROM user WHERE user="";

mysql> flush privileges;

**8 常见的数据库安全建议都有“降低给各方的特权”这一说法。**对于MySQL也是如此。

一般情况下，开发人员会使用最大的许可，不像安全管理一样考虑许可原则，而这样做会将数据库暴露在巨大的风险中。为保护数据库，务必保证真正存储MySQL数据库的文件目录是由”mysql” 用户和” mysql”组所拥有的。

shell>ls -l /var/lib/mysql

此外，还要确保仅有用户”mysql”和root用户可以访问/var/lib/mysql目录。

Mysql的二进制文件存在于/usr/bin/目录中，它应当由root用户或特定的”mysql”用户所拥有。对这些文件，其它用户不应当拥有“写”的访问权：

shell>ls -l /usr/bin/my\*

**9有些应用程序是通过一个特定数据库表的用户名和口令连接到MySQL的，安全人员不应当给予这个用户完全的访问权。**

如果攻击者获得了这个拥有完全访问权的用户，他也就拥有了所有的数据库。查看一个用户许可的方法是在MySQL控制台中使用命令SHOW GRANT

>SHOW GRANTS FOR 'user'@'localhost';

为定义用户的访问权，使用GRANT命令。在下面的例子中，user1仅能从dianshang数据库的billing表中选择：

> GRANT SELECT ON billing.dianshang TO 'user1'@'localhost';

> FLUSH PRIVILEGES;

如此一来，user1用户就无法改变数据库中这个表和其它表的任何数据。

另一方面，如果你要从一个用户移除访问权，就应使用一个与GRANT命令类似的REVOKE命令：

> REVOKE SELECT ON billing.ecommerce FROM 'user1'@'localhost';

> FLUSH PRIVILEGES;

**10 在用户访问MySQL控制台时，所有的命令历史都被记录在~/.mysql\_history中。如果攻击者访问这个文件，他就可以知道数据库的结构。**

$ cat ~/.mysql\_history

为了移除和禁用这个文件，应将日志发送到/dev/null。

$export MYSQL\_HISTFILE=/dev/null

上述命令使所有的日志文件都定向到/dev/null，你应当从home文件夹移除.mysql\_history：$ rm ~/.mysql\_history，并创建一个到/dev/null的符号链接。

**11 务必保持数据库为最新版本。因为攻击者可以利用上一个版本的已知漏洞来访问企业的数据库。**

**12 如果你的数据库服务器并不执行任何查询，建议你启用跟踪记录，你可以通过在/etc/my.cnf文件的[Mysql]部分添加：log =/var/log/mylogfile。**

对于生产环境中任务繁重的MySQL数据库，因为这会引起服务器的高昂成本。

此外，还要保证只有root和mysql可以访问这些日志文件。

**（1）错误日志**

务必确保只有root和mysql可以访问hostname.err日志文件。该文件存放在mysql数据历史中。该文件包含着非常敏感的信息，如口令、地址、表名、存储过程名、代码等，它可被用于信息收集，并且在某些情况下，还可以向攻击者提供利用数据库漏洞的信息。攻击者还可以知道安装数据库的机器或内部的数据。

**（2）MySQL日志**

确保只有root和mysql可以访问logfileXY日志文件，此文件存放在mysql的历史目录中。

## 十一、Mysql复制原理及其流程

**1 在Slave 服务器上执行sart slave命令开启主从复制开关，开始进行主从复制。**

**2 此时，Slave服务器的IO线程会通过在master上已经授权的复制用户权限请求连接master服务器**，并请求从执行binlog日志文件的指定位置（日志文件名和位置就是在配置主从复制服务时执行change master命令指定的）之后开始发送binlog日志内容

**3 Master服务器接收到来自Slave服务器的IO线程的请求后，二进制转储IO线程会根据**Slave服务器的IO线程请求的信息分批读取指定binlog日志文件指定位置之

后的binlog日志信息，然后返回给Slave端的IO线程。返回的信息中除了binlog日志内容外，还有在master服务器端记录的新的binlog文件名称，以及在新的binlog

中的下一个指定更新位置。

**4 当Slave服务**器的IO线程获取到Master服务器上IO线程发送的日志内容、日志文件及位置点后，会将binlog日志内容依次写到Slave端自身的Relay Log（即中继日志）文件（MySQL-relay-bin.xxx）的最末端，并将新的binlog文件名和位置记录到master-info文件中，以便下一次读取master端新binlog日志时能告诉Master服务器从新binlog日志的指定文件及位置开始读取新的binlog日志内容

**5 Slave服务器端的SQL线程会实时检测本地**Relay Log 中IO线程新增的日志内容，然后及时把**Relay LOG 文件中的内容解析成sql语句**，并在自身Slave服务器上按解析SQL语句的位置顺序执行应用这样sql语句，并在relay-log.info中记录当前应用中继日志的文件名和位置点

## 十二、Mysql的存储引擎

**1. MyISAM :** 是旧版本mysql的默认引擎，现在默认引擎是InnoDB。MyISAM引擎的主要特点就是快，没有事务处理操作，也不支持外键操作。适合于多读取插入，少更新删除的操作表。存储数据分成三个文件：.frm(存储表定义) .MYD(存储数据) .MYI(存储索引)

用法： engine=myisam default charset=utf-8 ;

**2.InnoDB** ：是新版本mysql的默认引擎，支持事务处理和外键，但是其缺点几就是慢了些。存储方式分为两种：1.共享表空间存储。[.frm(表结构) 和 innodb\_data\_home(数据)和innodb\_data\_file\_path(索引)] 2.多表空间存储。 [.frm(表结构) 和 .idb（数据） ]。

适用于对于事务由较高要求的表的创建。

用法：engine=innodb default charset=utf-8 ;

**3.MEMORY：** 数据访问非常快的引擎，存储格式同一放在内存中的一个磁盘文件中格式是.frm 。默认使用hash索引。一旦服务器关闭表中的数据就会丢失。数据大小有限制。

用法：engine=memory ;

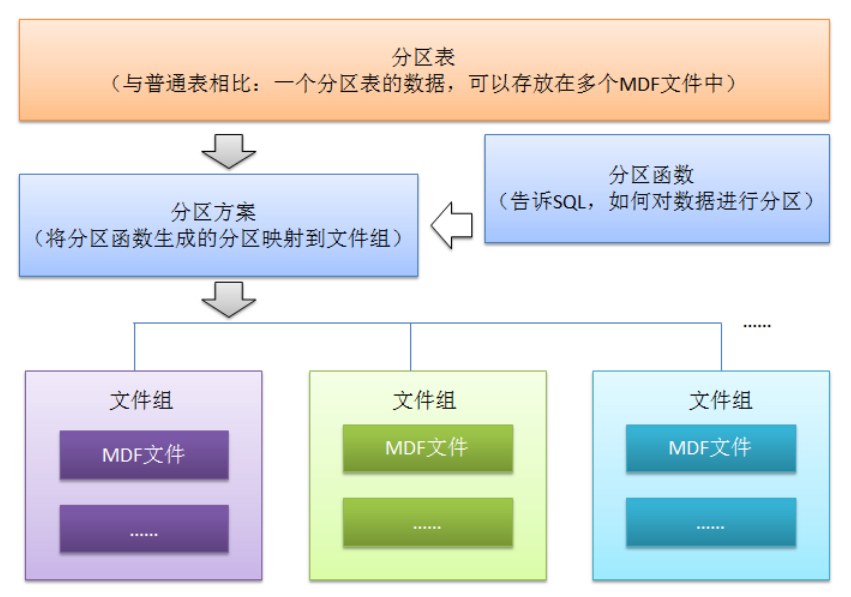
**4.MERGE：**本身是融合的意思，实质是MyISUM表的融合，这些融合的表必须结构完全相同。MERGE本身是没有数据的。插入操作可以是first或者是last。删除只是删除MERGE表定义，并不删除真正表的数据。存储方式：.frm(文件存储表定义信息) .MRG(描述组合表的信息，比如由哪些表组成，插入时的依据)。

适用于：将一系列等同的MyISAM表逻辑方式组合在一起，作为一个对象引用它们。

用法：engine=merge union=(\_\_,\_\_) insert\_method=last/first ;

## Mysql垂直分库/水平分表/水平分区/水平分片/读写分离/主从复制

**1水平分区，**就是将一个数据量比较大的表，用某种方法把数据从物理上分成若干个小表来存储，从逻辑来看还是一个大表。通俗地讲表分区是将一大表，根据条件分割成若干个小表。mysql5.1开始支持数据表分区了。 如：某用户表的记录超过了600万条，那么就可以根据入库日期将表分区，也可以根据所在地将表分区。当然也可根据其他的条件分区。



**（1）RANGE分区**

CREATE TABLE employees (

id INT NOT NULL,

fname VARCHAR(30),

lname VARCHAR(30),

hired DATE NOT NULL DEFAULT '1970-01-01',

separated DATE NOT NULL DEFAULT '9999-12-31',

job\_code INT NOT NULL,

store\_id INT NOT NULL

)

partition BY RANGE (store\_id) (

partition p0 VALUES LESS THAN (6),

partition p1 VALUES LESS THAN (11),

partition p2 VALUES LESS THAN (16),

partition p3 VALUES LESS THAN (21)

);

**（2）LIST分区**

类似于按RANGE分区，区别在于LIST分区是基于列值匹配一个离散值集合中的某个值来进行选择。

LIST分区通过使用“PARTITION BY LIST(expr)”来实现，其中“expr”是某列值或一个基于某个列值、并返回一个整数值的表达式，然后通过“VALUES IN (value\_list)”的方式来定义每个分区，其中“value\_list”是一个通过逗号分隔的整数列表。 注释：在MySQL 5.1中，当使用LIST分区时，有可能只能匹配整数列表。

CREATE TABLE employees (

id INT NOT NULL,

fname VARCHAR(30),

lname VARCHAR(30),

hired DATE NOT NULL DEFAULT '1970-01-01',

separated DATE NOT NULL DEFAULT '9999-12-31',

job\_code INT,

store\_id INT

)

PARTITION BY LIST(store\_id)

PARTITION pNorth VALUES IN (3,5,6,9,17),

PARTITION pEast VALUES IN (1,2,10,11,19,20),

PARTITION pWest VALUES IN (4,12,13,14,18),

PARTITION pCentral VALUES IN (7,8,15,16)

);

**（3）HASH分区**

基于用户定义的表达式的返回值来进行选择的分区，该表达式使用将要插入到表中的这些行的列值进行计算。这个函数可以包含MySQL 中有效的、产生非负整数值的任何表达式。

要使用HASH分区来分割一个表，要在CREATE TABLE 语句上添加一个“PARTITION BY HASH (expr)”子句，其中“expr”是一个返回一个整数的表达式。它可以仅仅是字段类型为MySQL整型的一列的名字。此外，你很可能需要在后面再添加一个“PARTITIONS num”子句，其中num是一个非负的整数，它表示表将要被分割成分区的数量。

CREATE TABLE employees (

id INT NOT NULL,

fname VARCHAR(30),

lname VARCHAR(30),

hired DATE NOT NULL DEFAULT '1970-01-01',

separated DATE NOT NULL DEFAULT '9999-12-31',

job\_code INT,

store\_id INT

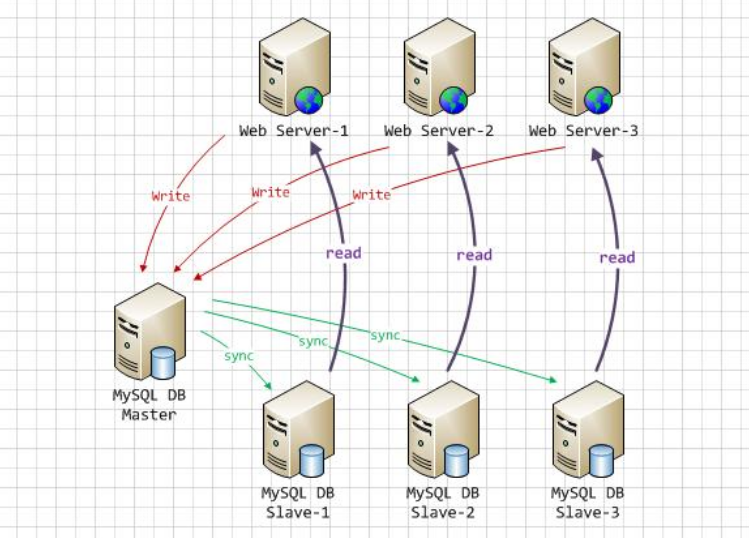
)

PARTITION BY HASH(store\_id)

PARTITIONS 4;

**2 读写分离&主从复制**

Mysql作为目前世界上使用最广泛的免费数据库，相信所有从事系统运维的工程师都一定接触过。但在实际的生产环境中，由单台Mysql作为独立的数据库是完全不能满足实际需求的，无论是在安全性，高可用性以及高并发等各个方面。因此，一般来说都是通过 主从复制（Master-Slave）的方式来同步数据，再通过读写分离（MySQL-Proxy）来提升数据库的并发负载能力 这样的方案来进行部署与实施的。



**（1）主服务器上进行的操作**

启动mysql服务

/opt/mysql/init.d/mysql start

通过命令行登录管理MySQL服务器

/opt/mysql/bin/mysql -uroot -p'new-password'

授权给从数据库服务器192.168.10.131

mysql> GRANT REPLICATION SLAVE ON \*.\* to 'rep1'@'192.168.10.131' identified by ‘password’;

**（2）配置从服务器**

修改从服务器的配置文件/opt/mysql/etc/my.cnf

将 server-id = 1修改为 server-id = 10，并确保这个ID没有被别的MySQL服务所使用。

启动mysql服务

/opt/mysql/init.d/mysql start

通过命令行登录管理MySQL服务器

/opt/mysql/bin/mysql -uroot -p'new-password'

执行同步SQL语句

mysql> change master to

master\_host=’192.168.10.130’,

master\_user=’rep1’,

master\_password=’password’,

master\_log\_file=’mysql-bin.000005’,

master\_log\_pos=261;

正确执行后启动Slave同步进程

mysql> start slave;

主从同步检查

mysql> show slave status\G

==============================================

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Slave\_IO\_State:

Master\_Host: 192.168.10.130

Master\_User: rep1

Master\_Port: 3306

Connect\_Retry: 60

Master\_Log\_File: mysql-bin.000005

Read\_Master\_Log\_Pos: 415

Relay\_Log\_File: localhost-relay-bin.000008

Relay\_Log\_Pos: 561

Relay\_Master\_Log\_File: mysql-bin.000005

Slave\_IO\_Running: YES

Slave\_SQL\_Running: YES

Replicate\_Do\_DB:

……………省略若干……………

Master\_Server\_Id: 1

1 row in set (0.01 sec)

==============================================

其中Slave\_IO\_Running 与 Slave\_SQL\_Running 的值都必须为YES，才表明状态正常。

**（3）验证主从复制效果**

主服务器上的操作

在主服务器上创建数据库first\_db

mysql> create database first\_db;

Query Ok, 1 row affected (0.01 sec)

在主服务器上创建表first\_tb

mysql> create table first\_tb(id int(3),name char(10));

Query Ok, 1 row affected (0.00 sec)

在主服务器上的表first\_tb中插入记录

mysql> insert into first\_tb values (001,’myself’);

Query Ok, 1 row affected (0.00 sec)

在从服务器上查看

mysql> show databases;

=============================

+--------------------+

| Database |

+--------------------+

| information\_schema |

| first\_db |

| mysql |

| performance\_schema |

| test |

+--------------------+

5 rows in set (0.01 sec)

=============================

数据库first\_db已经自动生成

mysql> use first\_db

Database chaged

mysql> show tables;

=============================

+--------------------+

| Tables\_in\_first\_db |

+--------------------+

| first\_tb |

+--------------------+

1 row in set (0.02 sec)

=============================

数据库表first\_tb也已经自动创建

mysql> select \* from first\_tb;

=============================

+------+------+

| id | name |

+------+------+

| 1 | myself |

+------+------+

1 rows in set (0.00 sec)

=============================

记录也已经存在

**（4）MySQL读写分离**

创建用于读写分离的数据库连接用户

登陆主数据库服务器192.168.10.130，通过命令行登录管理MySQL服务器

/opt/mysql/bin/mysql -uroot -p'new-password'

mysql> GRANT ALL ON \*.\* TO 'proxy1'@'192.168.10.132' IDENTIFIED BY 'password';

由于我们配置了主从复制功能，因此从数据库服务器192.168.10.131上已经同步了此操作。

为了清晰的看到读写分离的效果，需要暂时关闭MySQL主从复制功能

登陆从数据库服务器192.168.10.131，通过命令行登录管理MySQL服务器

/opt/mysql/bin/mysql -uroot -p'new-password'

关闭Slave同步进程

mysql> stop slave;

连接MySQL-Proxy

/opt/mysql/bin/mysql -uproxy1 -p'password' -P4040 -h192.168.10.132

登陆成功后，在first\_db数据的first\_tb表中插入两条记录

mysql> use first\_db;

Database changed

mysql> insert into first\_tb values (007,’first’);

Query Ok, 1 row affected (0.00 sec)

mysql> insert into first\_tb values (110,’second’);

Query Ok, 1 row affected (0.00 sec)

查询记录

mysql> select \* from first\_tb;

=============================

+------+------+

| id | name |

+------+------+

| 1 | myself |

+------+------+

1 rows in set (0.00 sec)

=============================

通过读操作并没有看到新记录

mysql> quit

退出MySQL-Proxy

下面，分别登陆到主从数据库服务器，对比记录信息

首先，检查主数据库服务器

mysql> select \* from first\_tb;

=============================

+------+------+

| id | name |

+------+------+

| 1 | myself |

+------+------+

| 007 | first |

+------+------+

| 110 | second |

+------+------+

3 rows in set (0.00 sec)

=============================

两条新记录都已经存在

然后，检查从数据库服务器

mysql> select \* from first\_tb;

=============================

+------+------+

| id | name |

+------+------+

| 1 | myself |

+------+------+

1 rows in set (0.00 sec)

=============================

没有新记录存在

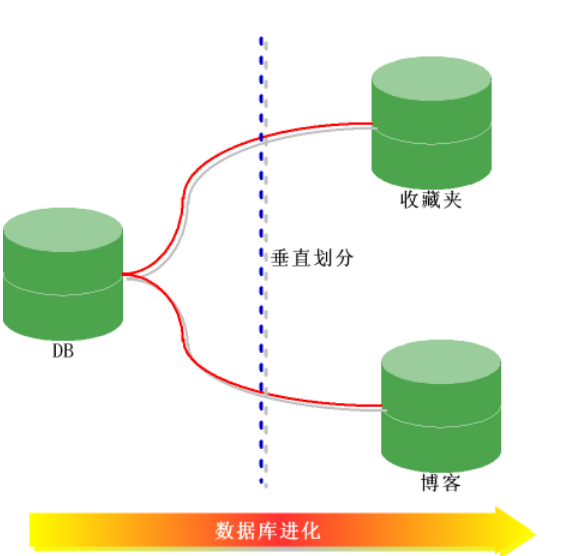
由此验证，我们已经实现了MySQL读写分离，目前所有的写操作都全部在Master主服务器上，用来避免数据的不同步；

另外，所有的读操作都分摊给了其它各个Slave从服务器上，用来分担数据库压力。

**3 垂直分库**

**按照功能划分，把数据分别放到不同的数据库和服务器。**

当一个网站开始刚刚创建时，可能只是考虑一天只有几十或者几百个人访问，数据库可能就个db，所有表都放一起，一台普通的服务器可能就够了，而且开发人员也非常高兴，而且信心十足，因为所有的表都在一个库中，这样查询语句就可以随便关联了，多美的一件事情。但是随着访问压力的增加，读写操作不断增加，数据库的压力绝对越来越大，可能接近极限，这时可能人们想到增加从服务器，做什么集群之类的，可是问题又来了，数据量也快速增长。这时可以考虑对读写操作进行分离，按照业务把不同的数据放到不同的库中。其实在一个大型而且臃肿的数据库中表和表之间的数据很多是没有关系的，或者更加不需要（join）操作，理论上就应该把他们分别放到不同的服务器。例如用户的收藏夹的数据和博客的数据库就可以放到两个独立的服务器。这个就叫垂直划分（其实叫什么不重要）。



**4 mysql分片与分区的区别**

MySQL5.1提供的分区(Partition)功能确实可以实现表的分区，但是这种分区是局限在单个数据库范围里的，它不能跨越服务器的限制。

**（一）单库单表：一个库，一个表T\_USER**

**（二）水平分区：就是将一个数据量比较大的表，用某种方法把数据从物理上分成若干个小表来存储，从逻辑来看还是一个大表。**

**（三）水平分表：可以通过某种方式将user进行水平的切分，产生两个表结构完全一样的user\_0000,user\_0001等表，user\_0000 + user\_0001 + …的数据刚好是一份完整的数据。**

**（四）水平分库：则把一个表的数据划分到不同的数据库，两个数据库的表结构一样。根据一定的规则，可以根据数据的产生者来做引导，上面的数据是由人产生的，可以根据人的id来划分数据库。然后再根据一定的规则，先获知数据在哪个数据库。**

**（五）垂直分库：按照业务不同，拆分成多个库 。**

**（六）读写分离/主从复制：一般来说都是通过 主从复制（Master-Slave）的方式来同步数据，再通过读写分离（MySQL-Proxy）来提升数据库的并发负载能力 这样的方案来进行部署与实施的。**

**（七）水平分片：在分布式存储系统中，数据需要分散存储在多台设备上，数据分片（Sharding）就是用来确定数据在多台存储设备上分布的技术。数据分片要达到三个目的：**

**分布均匀，即每台设备上的数据量要尽可能相近；负载均衡，即每台设备上的请求量要尽可能相近；扩缩容时产生的数据迁移尽可能少。**

**水平分区功能确实可以实现表的分区，但是这种分区是局限在单个数据库范围里的，它不能跨越服务器的限制。**

## 十五、Mysql并发/锁/事务

**1、什么是事务**

事务是一条或多条数据库操作语句的组合，具备ACID，4个特点。

原子性：要不全部成功，要不全部撤销

隔离性：事务之间相互独立，互不干扰

一致性：数据库正确地改变状态后，数据库的一致性约束没有被破坏

持久性：事务的提交结果，将持久保存在数据库中

**2、事务并发会产生什么问题**

**1）第一类丢失更新：**在没有事务隔离的情况下，两个事务都同时更新一行数据，但是第二个事务却中途失败退出， 导致对数据的两个修改都失效了。

例如：

张三的工资为5000，事务A中获取工资为5000，事务B获取工资为5000，汇入100，并提交数据库，工资变为5100，

随后

事务A发生异常，回滚了，恢复张三的工资为5000，这样就导致事务B的更新丢失了。

**2）脏读：**脏读就是指当一个事务正在访问数据，并且对数据进行了修改，而这种修改还没有提交到数据库中，这时，另外一个事务也访问这个数据，然后使用了这个数据。

例如：

　　张三的工资为5000,事务A中把他的工资改为8000,但事务A尚未提交。

　　与此同时，

　　事务B正在读取张三的工资，读取到张三的工资为8000。

　　随后，

　　事务A发生异常，而回滚了事务。张三的工资又回滚为5000。

　　最后，

　　事务B读取到的张三工资为8000的数据即为脏数据，事务B做了一次脏读。

**3）不可重复读：**是指在一个事务内，多次读同一数据。在这个事务还没有结束时，另外一个事务也访问该同一数据。那么，在第一个事务中的两次读数据之间，由于第二个事务的修改，那么第一个事务两次读到的的数据可能是不一样的。这样就发生了在一个事务内两次读到的数据是不一样的，因此称为是不可重复读。

例如：

　　在事务A中，读取到张三的工资为5000，操作没有完成，事务还没提交。

　　与此同时，

　　事务B把张三的工资改为8000，并提交了事务。

　　随后，

　　在事务A中，再次读取张三的工资，此时工资变为8000。在一个事务中前后两次读取的结果并不致，导致了不可重复读。

**4）第二类丢失更新：**不可重复读的特例。有两个并发事务同时读取同一行数据，然后其中一个对它进行修改提交，而另一个也进行了修改提交。这就会造成第一次写操作失效。

例如：

在事务A中，读取到张三的存款为5000，操作没有完成，事务还没提交。

　　与此同时，

　　事务B，存储1000，把张三的存款改为6000，并提交了事务。

　　随后，

　　在事务A中，存储500，把张三的存款改为5500，并提交了事务，这样事务A的更新覆盖了事务B的更新。

**5）幻读：**是指当事务不是独立执行时发生的一种现象，例如第一个事务对一个表中的数据进行了修改，这种修改涉及到表中的全部数据行。同时，第二个事务也修改这个表中的数据，这种修改是向表中插入一行新数据。那么，以后就会发生操作第一个事务的用户发现表中还有没有修改的数据行，就好象发生了幻觉一样。

例如：

　　目前工资为5000的员工有10人，事务A读取所有工资为5000的人数为10人。

　　此时，

　　事务B插入一条工资也为5000的记录。

　　这是，事务A再次读取工资为5000的员工，记录为11人。此时产生了幻读。

提醒：

不可重复读的重点是修改，同样的条件，你读取过的数据，再次读取出来发现值不一样了

幻读的重点在于新增或者删除，同样的条件，第 1 次和第 2 次读出来的记录数不一样

**3、事务隔离级别，解决什么并发问题，以及存在什么并发问题**

（1）READ\_UNCOMMITTED

　　这是事务最低的隔离级别，它充许另外一个事务可以看到这个事务未提交的数据。

　　解决第一类丢失更新的问题，但是会出现脏读、不可重复读、第二类丢失更新的问题，幻读 。

（2）READ\_COMMITTED

　　保证一个事务修改的数据提交后才能被另外一个事务读取，即另外一个事务不能读取该事务未提交的数据。

　　解决第一类丢失更新和脏读的问题，但会出现不可重复读、第二类丢失更新的问题，幻读问题

（3）REPEATABLE\_READ

　　保证一个事务相同条件下前后两次获取的数据是一致的

解决第一类丢失更新，脏读、不可重复读、第二类丢失更新的问题，但会出幻读。

（4）SERIALIZABLE

　　事务被处理为顺序执行。

　　解决所有问题

提醒：

**Mysql默认的事务隔离级别为repeatable\_read**

**4、InnoDB引擎的锁机制（之所以以InnoDB为主介绍锁，是因为InnoDB支持事务，支持行锁和表锁用的比较多，Myisam不支持事务，只支持表锁）**

（1）共享锁（S）：允许一个事务去读一行，阻止其他事务获得相同数据集的排他锁。

（2）排他锁（X)：允许获得排他锁的事务更新数据，阻止其他事务取得相同数据集的共享读锁和排他写锁。

（3）意向共享锁（IS）：事务打算给数据行加行共享锁，事务在给一个数据行加共享锁前必须先取得该表的IS锁。

（4）意向排他锁（IX）：事务打算给数据行加行排他锁，事务在给一个数据行加排他锁前必须先取得该表的IX锁。

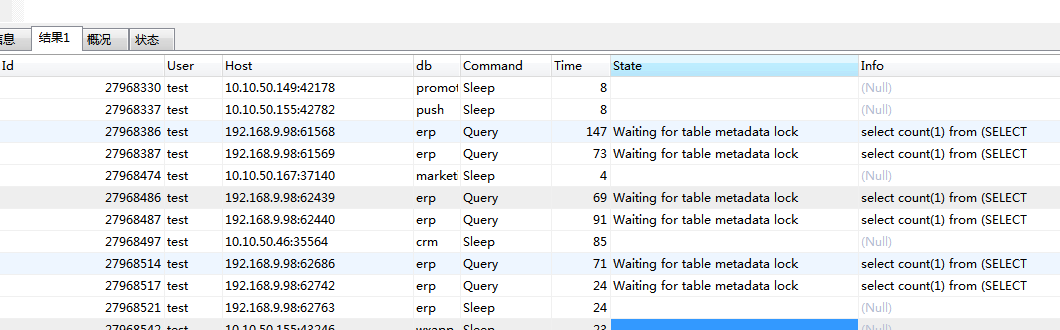
**共享锁【S锁】**  
又称读锁，若事务T对数据对象A加上S锁，则事务T可以读A但不能修改A，其他事务只能再对A加S锁，而不能加X锁，直到T释放A上的S锁。这保证了其他事务可以读A，但在T释放A上的S锁之前不能对A做任何修改。

**排他锁【X锁】**  
又称写锁。若事务T对数据对象A加上X锁，事务T可以读A也可以修改A，其他事务不能再对A加任何锁，直到T释放A上的锁。这保证了其他事务在T释放A上的锁之前不能再读取和修改A。

# waiting for table metadata lock 问题深入分析

一个没提交的事务使用了A表， 另外一个session 对A表进行alter，出现waiting for table metadata lock   
  
在insert into t select \* from share 运行时， 同时执行alter table t add index(play\_count)，   
alter table语句会Waiting for table metadata lock， 直到insert into … select 语句结束。

1. **>show processlist ; 找到线程ID和库**



**2>>kill 279685174**



**3>>解决**

**5 说明**

1）共享锁和排他锁都是行锁，意向锁都是表锁，应用中我们只会使用到共享锁和排他锁，意向锁是mysql内部使用的，不需要用户干预。

2）对于UPDATE、DELETE和INSERT语句，InnoDB会自动给涉及数据集加排他锁（X)；对于普通SELECT语句，InnoDB不会加任何锁，事务可以通过以下语句显示给记录集加共享锁或排他锁。

共享锁（S）：SELECT \* FROM table\_name WHERE ... LOCK IN SHARE MODE。

排他锁（X)：SELECT \* FROM table\_name WHERE ... FOR UPDATE。

3）InnoDB行锁是通过给索引上的索引项加锁来实现的，因此InnoDB这种行锁实现特点意味着：只有通过索引条件检索数据，InnoDB才使用行级锁，否则，InnoDB将使用表锁！。

## 十六、Mysql系统表

**1.    获取所有表结构(TABLES)**

**SELECT  \*  FROM information\_schema.TABLES WHERE  TABLE\_SCHEMA='数据库名';**

TABLES表：提供了关于数据库中的表的信息（包括视图）。详细表述了某个表属于哪个schema，表类型，表引擎，创建时间等信息。各字段说明如下：

**字段 含义**

Table\_catalog 数据表登记目录

Table\_schema 数据表所属的数据库名

Table\_name 表名称

Table\_type 表类型[system view|base table]

Engine 使用的数据库引擎[MyISAM|CSV|InnoDB]

Version 版本，默认值10

Row\_format 行格式[Compact|Dynamic|Fixed]

Table\_rows 表里所存多少行数据

Avg\_row\_length 平均行长度

Data\_length 数据长度

Max\_data\_length 最大数据长度

Index\_length 索引长度

Data\_free 自由数据？

Auto\_increment 做自增主键的自动增量当前值

Create\_time 表的创建时间

Update\_time 表的更新时间

Check\_time 表的检查时间

Table\_collation 表的字符校验编码集

Checksum 校验和

Create\_options 创建选项

Table\_comment 表的注释、备注

**2.    获取表字段(COLUMNS)**

**SELECT \* FROM information\_schema.COLUMNS WHERE TABLE\_SCHEMA='数据库名' AND TABLE\_NAME='表名'**

COLUMNS表：提供了表中的列信息。详细表述了某张表的所有列以及每个列的信息。各字段的说明信息如下：

**字段 含义**

Table\_catalog 数据表登记目录

Table\_schema 数据表所属的数据库名

Table\_name 所属的表名称

Column\_name 列名称

Ordinal\_position 字段在表中第几列

Column\_default 列的默认数据

Is\_nullable 字段是否可以为空

Data\_type 数据类型

Character\_maximum\_length 字符最大长度

Character\_octet\_length 字节长度？

Numeric\_precision 数据精度

Numeric\_scale 数据规模

Character\_set\_name 字符集名称

Collation\_name 字符集校验名称

Column\_type 列类型

Column\_key 关键列[NULL|MUL|PRI]

Extra 额外描述[NULL|on update CURRENT\_TIMESTAMP|auto\_increment]

Privileges 字段操作权限[select|select,insert,update,references]

Column\_comment 字段注释、描述

**3 获取表键值**

**SELECT  \*  FROM information\_schema.KEY\_COLUMN\_USAGE  WHERE  TABLE\_SCHEMA='数据库名'  AND TABLE\_NAME='表名'**

KEY\_COLUMN\_USAGE表：存取表的健值。各字段的说明信息如下：

字段 含义

Constraint\_catalog 约束登记目录

Constraint\_schema 约束所属的数据库名

Constraint\_name 约束的名称

Table\_catalog 数据表等级目录

Table\_schema 键值所属表所属的数据库名（一般与Constraint\_schema值相同）

Table\_name 键值所属的表名

Column\_name 键值所属的列名

Ordinal\_position 键值所属的字段在表中第几列

Position\_in\_unique\_constraint 键值所属的字段在唯一约束的位置（若为外键值为1）

Referenced\_talble\_schema 外键依赖的数据库名（一般与Constraint\_schema值相同）

Referenced\_talble\_name 外键依赖的表名

Referenced\_column\_name 外键依赖的列名

**4.    获取表Check约束**

**SELECT \* FROM information\_schema.TABLE\_CONSTRAINTS  WHERE  TABLE\_SCHEMA='数据库名' AND TABLE\_NAME='表名'**

TABLE\_CONSTRAINTS表：存储主键约束、外键约束、唯一约束、check约束。各字段的

字段 含义

Constraint\_catalog 约束登记目录

Constraint\_schema 约束所属的数据库名

Constraint\_name 约束的名称

Table\_schema 约束依赖表所属的数据库名（一般与Constraint\_schema值相同）

Table\_name 约束所属的表名

Constraint\_type 约束类型[primary key|foreign key|unique|check]

**5.    获取表索引**

**SELECT \* FROM information\_schema.STATISTICS WHERE TABLE\_SCHEMA='数据库名' AND TABLE\_NAME='表名'**

STATISTICS表：提供了关于表索引的信息。各字段的说明信息如下：

字段 含义

Table\_catalog 数据表登记目录

Table\_schema 索引所属表的数据库名

Table\_name 索引所属的表名

Non\_unique 字段不唯一的标识

Index\_schema 索引所属的数据库名（一般与table\_schema值相同）

Index\_name 索引名称

Seq\_in\_index

Column\_name 索引列的列名

Collation 校对，列值全显示为A

Cardinality 基数（一般与该表的数据行数相同）

Sub\_part

Packed 是否包装过，默认为NULL

Nullable 是否为空[‘’|YES|NO]

Index\_type 索引的类型，列值全显示为BTREE（平衡树索引）

Comment 索引注释、备注

**6.    mysql有关show的用法**

SHOW DATABASES列出 MySQL Server上的数据库。

SHOW TABLES [FROM db\_name]列出数据库中的表。

SHOW TABLE STATUS [FROM db\_name]列出数据库的表信息，比较详细。

SHOW COLUMNS FROM tbl\_name [FROM db\_name]列出表的列信息，同 SHOW FIELDS FROM tbl\_name [FROM db\_name]，DESCRIBE tbl\_name [col\_name]。

SHOW FULL COLUMNS FROM tbl\_name [FROM db\_name]列出表的列信息，比较详细，同 SHOW FULL FIELDS FROM tbl\_name [FROM db\_name]。

SHOW INDEX FROM tbl\_name [FROM db\_name]列出表的索引信息。

SHOW STATUS列出 Server 的状态信息。

SHOW VARIABLES列出 MySQL 系参数值

SHOW PROCESSLIST查看当前mysql查询进程

SHOW GRANTS FOR user列出用户的授权命令

## 十七、Mysql内核

**1 Mysql体系结构**

MySQL是一个可移植的数据库，几乎能在当前所有的操作系统上运行，如Unix/Linux、Windows、Mac和Solaris。各种系统在底层实现方面各有不同，但是MySQL基本上能保证在各个平台上的物理体系结构的一致性。

**2 MySQL由以下几部分组成：**

- 连接池组件

- 管理服务和工具组件

- 查询分析器组件

- 优化器组件

- 缓冲组件

- 插件式存储引擎

- 物理文件

Connection:不同语言与SQL的交互

Management Serveices &Utilities:系统管理和控制工具—备份和恢复的安全性，复制，集群，管理，配置，迁移和元数据。

Connection Pool:连接池—进行身份验证，线程重用，连接限制，检查内存，数据缓冲；管理用户的链接，线程处理等需要缓冲的需求。

SQL Interface:SQL接口—进行DML,DDL，存储过程，视图，触发器等操作和管理；用户通过SQL命令来查询所需要结果。

Parser:解析器—查询翻译对象的特权；SQL命令传递到解析器的时候会被解析器验证和解析。

Optimize:查询优化器—访问路径的统计数据；

Cache和Buffer:查询缓冲器—全局和引擎特定的缓冲和缓冲区；

Pluggable Storage Engine:插件式存储引擎；

**3 Mysql源码组织结构**

build:内含在各个平台，各种编译器下进行编译的脚本。如compile-pentium-debug表示在pentium架构上进行编译的脚本。

client:客户端工具，入mysql,mysqladmin之类。

Cmd-line-utils:readline,libedit工具。

Config:给aclocal使用配置文件。

Dbug:提供一些调试用的宏定义。

Extra:提供innochecksum,resolveip等额外的小工具。

Include:包含的头文件

Libmysql:库文件，生产libmysqlclient.so.

Libmysql\_r:线程安全地库文件，生成libmysqlclient\_r.so.

Libservices:5.5.0中新加的目录，实现了打印功能。

Man:手册页。

Mysql-test:mysqld的测试工具一套。

Mysys:为跨平台设计，MySQL自己实现了一套常用的数据结构和算法，如string,hash等。

Netware:在netwate平台上进行编译时需要的工具和库。

Plugin:mysql以插件形式实现的部分功能。

Pstack:异步栈追踪工具。

Regex:正则表示式工具。

Script:提供脚本工具，如mysql\_install\_db等。

Sql:mysql主要代码，将会生成mysqld文件。

Sql-bench:一些评测代码。

Sql-common:存放部分服务器端和客户端都会用到的代码。

Storage:存储引擎所在目录，如myisam,innodb,ndb等。

String:string库。

Support-files:my.cnf示例配置文件。

Tests:测试文件所在目录。

Unittest:单元测试。

Vio:virtual io 系统，是对network io的封装。

Win:给windows平台提供的编译环境

Zip:zip库工具。

**4 MySQL数据结构**

（1）THD线程描述符(sql/sql\_class.h)—包含处理用户请求时需要的相关数据，每个连接会有一个线程来处理，有一些高层函数中，次数据结构常被当作第一个参数传递。THD类中包含线程的许多用户信息。处理客户端的连接不是线程的唯一用途，MySQl还有一些系统线程，如从服务器复制进程和信号处理线程。

（2）NET网络连接描述符(sql/mysql\_com.h)–网络连接描述符，对内部数据包进行了封装，是client和server之间的通信协议。

（3）Item SQL表达式–Item用于实现表达式，如查询条目，函数，where,order,group,on子句等。每个Item类的实例都包含：a sql语句中某项的类似物，如where.b 一个值。c 数据类型描述符只要mysql与sql语句打交道，在赋值，比较，聚合和验证一个sql元素时，你就会发现mysql在使用item类与它的子类。

（4）TABLE数据库表描述符(sql/table.h)–表定义了MySQL数据库表在MySQL内存中的形式。表以两种形式存在–打开或者关闭状态。MySQL要想使用表，必须先将其打开。表描述类是在被打开时创建。表描述类创建后被置于表缓冲(table\_cache)中，等待下一个查询的重用，以提高访问的效率。数据库表描述符，，分成TABLE和TABLE\_SHARE两部分。

（5）FIELD字段描述符(sql/field.h)–域描述符，对列数据类型以及属性的定义，是各种字段的抽象基类。

**5 MySQL算法**

（1）排序算法:MySQL排序采用sort\_buffer\_size设定的线程独占内存进行排序，如果缓冲内放下要排序的数据，那么MySQL会采取载入所有要返回的值进入缓冲排序；如果缓冲放不下，则把要排序的列放入缓冲排序，然后再回表查询返回结果，如果依然放不下，就会采用归并排序，排好序的部分放入磁盘，载入未排好序的部分继续在内存中排序，然后和磁盘上的部分二路归并，知道排序全部完成。

（2）连接算法：MySQL关联采用join\_buffer\_size设定的线程独占内存进行join操作，join过程会采用下推算法，如果有where条件，mysql会进行过滤，然后采用关联字段去匹配驱动表，这样可以有效减少参与关联的记录数。

（3）缓冲算法：MySQL缓冲采用LRU算法，由系统维护一个LRU双向链表。

**6 MySQL主要执行过程**

从架构上来看，Mysql服务器对于一条SQL语句的执行过程可以分成如下几部分：

接受命令 包括用户验证，资源申请等

|

V

命令解析 解析SQL语句，生成语法树

|

V

寻找执行计划 根据解析出来的语法树，找到可能的执行计划。对于一条SQL语句，很可能会有多种执行方案，特别是在SQL语句比较复杂的时候。这里需要对于各种可能的方案进行代价评估，最快的找到最有的执行方案。

|

V

优化执行计划 优化执行计划。这是SQL执行中最复杂的部分之一，据说全都是由数学博士们写出来的，而且比较难懂。我目前还处于不懂的状态。

|

V

执行 没啥可说的，只剩执行及返回结果了

## 十八、Mysql服务器的优化

**1 存储引擎**

mysql中有多种存储引擎，一般常见的有三种

**MyIsam InnoDB Memory**

**用途** 快读 完整的事务支持 内存数据

**锁** 全表锁定 多种隔离级别的行锁 全表锁定

**持久性** 基于表恢复 基于日志的恢复 无磁盘I/O,无可持久性

**事务特性** 不支持 支持 不支持

**支持索引类型** B-tree/FullText/R-tree Hash/B-tree Hash/B-tree

通常的观点是MyISAM 注重性能，InnoDB注重事务，所以一边使用MyISAM类的表做非事务型的业务。这种观点是由于早期InnoDB引擎还不成熟，而事实上并不是这样的。mysql在高并发下的性能瓶颈很明显，主要原因就是锁定机制导致的堵塞。而InnoDB在锁定机制上采用了行级锁，不同于MyISAM的表级锁，行级锁在锁定上带来的消耗大于表级锁，但是在系统并发访问较高时，InnoDB整体的性能高于MyISAM。同时InnoDB的索引不仅缓存索引本身，也缓存数据，说以InnoDB需要更大的内存，在现在来说内存很廉价。而且在MYSQL8.0 中 MyISAM已经不存在了。

**2 mysql服务器调整优化**

关闭不必要的二进制日志和慢查询日志，仅在内存足够或开发调试时打开。使用下来语句查

看是否打开：

show variables like '%slow%';

使用下列语句查看慢查询条数：

show global status like '%slow%';

慢查询会消耗过度消耗CPU，可以间歇性打开慢查询日志来定位新能瓶颈。

增加mysql允许的最大连接数。查看mysql最大连接数：

show variables like 'max\_connections';

对于InnoDB存储引擎，需要安装服务器内存来设置innodb\_buffer\_pool\_size一般是操作系统内存的70%-80%最佳。

定期的执行optimize table tableName 进行碎片整理 ，或者在大量删除后进行。

对数据库进行分区，分表操作。

使用Nosql辅助,如：Memcached，redis。

**3 mysql最大连接数修改方法**

**方法一：进入MYSQL安装目录 打开MYSQL配置文件 my.ini** 或 my.cnf查找 max\_connections=100   修改为 max\_connections=（数字） 服务里重起MYSQL即可

**方法二：set GLOBAL max\_connections=200**

mysql修改innodb\_buffer\_pool\_size

进入MYSQL安装目录 打开MYSQL配置文件 my.ini 或 my.cnf 查找  innodb\_buffer\_pool\_size  这一项设置 修改为 例如：   innodb\_buffer\_pool\_size = 6G

增加mysql配置中的buffer和cache的数值

## 十九、Mysql各种参数常量设定

**1.系统mysql的进程数**

ps -ef | grep "mysql" | grep -v "grep" | wc –l

**2.Slave\_running**

mysql > show status like 'Slave\_running';

如果系统有一个从复制服务器，这个值指明了从服务器的健康度

**3.Threads\_connected**

mysql > show status like 'Threads\_connected';

当前客户端已连接的数量。这个值会少于预设的值，但你也能监视到这个值较大，这可保证客户端是处在活跃状态。

**4.Threads\_running**

mysql > show status like 'Threads\_running';

如果数据库超负荷了，你将会得到一个正在（查询的语句持续）增长的数值。这个值也可以少于预先设定的值。这个值在很短的时间内超过限定值是没问题的。当Threads\_running值超过预设值时并且该值在5秒内没有回落时， 要同时监视其他的一些值。

**5.Aborted\_clients**

mysql > show status like 'Aborted\_clients';

客户端被异常中断的数值，即连接到mysql服务器的客户端没有正常地断开或关闭。对于一些应用程序是没有影响的，但对于另一些应用程序可能你要跟踪该值，因为异常中断连接可能表明了一些应用程序有问题。

**6.Questions**

mysql> show status like 'Questions';

每秒钟获得的查询数量，也可以是全部查询的数量，根据你输入不同的命令会得到你想要的不同的值。

**7.Handler\_\***

mysql> show status like 'Handler\_%';

如果你想监视底层（low-level）数据库负载，这些值是值得去跟踪的。

如果Handler\_read\_rnd\_next值相对于你认为是正常值相差悬殊，可能会告诉你需要优化或索引出问题了。Handler\_rollback表明事务被回滚的查询数量。你可能想调查一下原因。

**8.Opened\_tables**

mysql> show status like 'Opened\_tables';

表缓存没有命中的数量。如果该值很大，你可能需要增加table\_cache的数值。典型地，你可能想要这个值每秒打开的表数量少于1或2。

**9.Select\_full\_join**

mysql> show status like 'Select\_full\_join';

没有主键（key）联合（Join）的执行。该值可能是零。这是捕获开发错误的好方法，因为一些这样的查询可能降低系统的性能。

**10.Select\_scan**

mysql> show status like 'Select\_scan';

执行全表搜索查询的数量。在某些情况下是没问题的，但占总查询数量该比值应该是常量（即Select\_scan/总查询数量商应该是常数）。如果你发现该值持续增长，说明需要优化，缺乏必要的索引或其他问题。

**11.Slow\_queries**

mysql> show status like 'Slow\_queries';

超过该值（--long-query-time）的查询数量，或没有使用索引查询数量。对于全部查询会有小的冲突。如果该值增长，表明系统有性能问题。

**12.Threads\_created**

mysql> show status like 'Threads\_created';

该值应该是低的。较高的值可能意味着你需要增加thread\_cache的数值，或你遇到了持续增加的连接，表明了潜在的问题。

**13.客户端连接进程数**

shell> mysqladmin processlist

mysql> show processlist;

你可以通过使用其他的统计信息得到已连接线程数量和正在运行线程的数量，检查正在运行的查询花了多长时间是一个好主意。如果有一些长时间的查询，管理员可以被通知。你可能也想了解多少个查询是在"Locked"的状态—---该值作为正在运行的查询不被计算在内而是作为非活跃的。一个用户正在等待一个数据库响应。

**14.innodb状态**

mysql> show innodb status;

该语句产生很多信息，从中你可以得到你感兴趣的。首先你要检查的就是“从最近的XX秒计算出来的每秒的平均负载”。

(1)Pending normal aio reads: 该值是innodb io请求查询的大小（size）。如果该值大到超过了10—20，你可能有一些瓶颈。

(2)reads/s, avg bytes/read, writes/s, fsyncs/s:这些值是io统计。对于reads/writes大值意味着io子系统正在被装载。适当的值取决于你系统的配置。

(3)Buffer pool hit rate:这个命中率非常依赖于你的应用程序。当你觉得有问题时请检查你的命中率。

(4)inserts/s, updates/s, deletes/s, reads/s:有一些Innodb的底层操作。你可以用这些值检查你的负载情况查看是否是期待的数值范围。

**15.主机性能状态**

shell> uptime

**16.CPU使用率**

shell> top

shell> vmstat

**17.磁盘IO**

shell> vmstat

shell> iostat

**18.swap进出量(内存)**

shell> free

**19.MySQL错误日志**

在服务器正常完成初始化后，什么都不会写到错误日志中，因此任何在该日志中的信息都要引起管理员的注意。

**20.InnoDB表空间信息**

InnoDB仅有的危险情况就是表空间填满----日志不会填满。检查的最好方式就是：show table status;你可以用任何InnoDB表来监视InnoDB表的剩余空间。

**21.QPS每秒Query量**

QPS = Questions(or Queries) / seconds

mysql > show /\* global \*/ status like 'Question';

**22.TPS(每秒事务量)**

TPS = (Com\_commit + Com\_rollback) / seconds

mysql > show status like 'Com\_commit';

mysql > show status like 'Com\_rollback';

**23.key Buffer 命中率**

key\_buffer\_read\_hits = (1-key\_reads / key\_read\_requests) \* 100%

key\_buffer\_write\_hits = (1-key\_writes / key\_write\_requests) \* 100%

mysql> show status like 'Key%';

**24.InnoDB Buffer命中率**

Innodb\_buffer\_read\_hits = (1 - innodb\_buffer\_pool\_reads / innodb\_buffer\_pool\_read\_requests) \* 100%

mysql> show status like 'innodb\_buffer\_pool\_read%';

**25.Query Cache命中率**

Query\_cache\_hits = (Qcahce\_hits / (Qcache\_hits + Qcache\_inserts )) \* 100%;

mysql> show status like 'Qcache%';

**26.Table Cache状态量**

mysql> show status like 'open%';

**27.Thread Cache 命中率**

Thread\_cache\_hits = (1 - Threads\_created / connections ) \* 100%

mysql> show status like 'Thread%';

mysql> show status like 'Connections';

**28.锁定状态**

mysql> show status like '%lock%';

**29.复制延时量**

mysql > show slave status

**30.Tmp Table状况(临时表状况)**

mysql > show status like 'Create\_tmp%';

**31.Binlog Cache使用状况**

mysql > show status like 'Binlog\_cache%';

**32.Innodb\_log\_waits量**

mysql > show status like 'innodb\_log\_waits

## 二十、容灾备份

**1 使用mysqldump命令备份**

mysqldump命令将数据库中的数据备份成一个文本文件。表的结构和表中的数据将存储在生成的文本文件中。

mysqldump命令的工作原理很简单。它先查出需要备份的表的结构，再在文本文件中生成一个CREATE语句。然后，将表中的所有记录转换成一条INSERT语句。然后通过这些语句，就能够创建表并插入数据。

**2 备份一个数据库**

　　mysqldump基本语法：

　　mysqldump -u username -p dbname table1 table2 ...-> BackupName.sql

　　其中：

dbname参数表示数据库的名称；

table1和table2参数表示需要备份的表的名称，为空则整个数据库备份；

BackupName.sql参数表设计备份文件的名称，文件名前面可以加上一个绝对路径。通常将数据库被分成一个后缀名为sql的文件；

　　使用root用户备份test数据库下的person表

mysqldump -u root -p test person > D:\backup.sql

　　其生成的脚本如下：

　　文件的开头会记录MySQL的版本、备份的主机名和数据库名。

　　文件中以“--”开头的都是SQL语言的注释，以"/\*!40101"等形式开头的是与MySQL有关的注释。40101是MySQL数据库的版本号，如果MySQL的版本比1.11高，则/\*!40101和\*/之间的内容就被当做SQL命令来执行，如果比4.1.1低就会被当做注释。

**3 备份多个数据库**

　　语法：

mysqldump -u username -p --databases dbname2 dbname2 > Backup.sql

　　加上了--databases选项，然后后面跟多个数据库

mysqldump -u root -p --databases test mysql > D:\backup.sql

**4 备份所有数据库**

　　mysqldump命令备份所有数据库的语法如下：

mysqldump -u username -p -all-databases > BackupName.sql

　　示例：

mysqldump -u -root -p -all-databases > D:\all.sql

**5 直接复制整个数据库目录**

　　MySQL有一种非常简单的备份方法，就是将MySQL中的数据库文件直接复制出来。这是最简单，速度最快的方法。不过在此之前，要先将服务器停止，这样才可以保证在复制期间数据库的数据不会发生变化。如果在复制数据库的过程中还有数据写入，就会造成数据不一致。这种情况在开发环境可以，但是在生产环境中很难允许备份服务器。

　　注意：这种方法不适用于InnoDB存储引擎的表，而对于MyISAM存储引擎的表很方便。同时，还原时MySQL的版本最好相同。

**6 使用mysqlhotcopy工具快速备份**

　　一看名字就知道是热备份。因此,mysqlhotcopy支持不停止MySQL服务器备份。而且，mysqlhotcopy的备份方式比mysqldump快。mysqlhotcopy是一个perl脚本，主要在Linux系统下使用。其使用LOCK TABLES、FLUSH TABLES和cp来进行快速备份。

　　原理：先将需要备份的数据库加上一个读锁，然后用FLUSH TABLES将内存中的数据写回到硬盘上的数据库，最后，把需要备份的数据库文件复制到目标目录。

　　命令格式如下：

[root@localhost ~]# mysqlhotcopy [option] dbname1 dbname2 backupDir/

dbname：数据库名称；

backupDir：备份到哪个文件夹下；

　　常用选项：

--help：查看mysqlhotcopy帮助；

--allowold：如果备份目录下存在相同的备份文件，将旧的备份文件加上\_old；

--keepold：如果备份目录下存在相同的备份文件，不删除旧的备份文件，而是将旧的文件更名；

--flushlog：本次辈分之后，将对数据库的更新记录到日志中；

--noindices：只备份数据文件，不备份索引文件；

--user=用户名：用来指定用户名，可以用-u代替；

--password=密码：用来指定密码，可以用-p代替。使用-p时，密码与-p之间没有空格；

--port=端口号：用来指定访问端口，可以用-P代替；

--socket=socket文件：用来指定socket文件，可以用-S代替；

　　mysqlhotcopy并非mysql自带，需要安装Perl的数据库接口包；下载地址为:http://dev.mysql.com/downloads/dbi.html

　　目前，该工具也仅仅能够备份MyISAM类型的表。

**7 数据还原**

（1）还原使用mysqldump命令备份的数据库的语法如下：

　　mysql -u root -p [dbname] < backup.sq

　　示例：

mysql -u root -p < C:\backup.sql

可备份数据

mysql -u root -p > C:\backup.sql

（2）还原直接复制目录的备份

通过这种方式还原时，必须保证两个MySQL数据库的版本号是相同的。MyISAM类型的表有效，对于InnoDB类型的表不可用，InnoDB表的表空间不能直接复制。

## 二十二、Mysql运行原理和架构

**1 体系架构**

**1).最上层：**

最上层是一些客户端和连接服务，包含本地的sock通信和大多数基于客户端/服务端工具实现的类似于tcp/ip的通信，主要完成一些类似于连接处理、授权认证及相关的安全方案，在该层上引用了线程池的概念，为通过认证安全接入的客户端提供线程。同样在该层上可以实现基于ssl的安全链接。服务器也会为安全接入的每个客户端验证它所具有的操作权限。

**2).第二层：**

第二层架构主要完成大多数的核心服务功能。如sql接口，并完成缓存的查询。sql的分析和优化 以及部分内置函数的执行。所有跨存储引擎的功能也在这一层实现，如过程，函数等。在该层，服务器会解析查询并创建相应的内部解析树，并对其完成相应的优化如确定查询表的顺序，是否利用索引等。最后生成相应的执行操作。如select语句，服务器还会查询内部的缓存。如果缓存空间足够大，这样就解决大量读操作的环境中能够很好的提升系统的性能。

**3).存储引擎层：**

存储引擎真正的负责MySQL中数据的存储和提取，服务器通过API与存储引擎进行通信，不同的存储引擎具有的功能不同，这样我们可以根据自己的实际需进行选取。

**4).数据存储层：**

主要是将数据存储在运行于裸设备的文件系统之上，并完成于存储引擎的交互。

**2 并发控制和锁的概念**

当数据库中有多个操作需要修改同一数据时，不可避免的会产生数据的脏读。这时就需要数据库具有良好的并发控制能力，这一切在MySQL中都是由服务器和存储引擎来实现的。解决并发问题最有效的方案是引入了锁的机制，锁在功能上分为共享锁(shared lock)和排它锁(exclusive lock)即通常说的读锁和写锁。当一个select语句在执行时可以施加读锁，这样就可以允许其它的select操作进行，因为在这个过程中数据信息是不会被改变的这样就能够提高数据库的运行效率。当需要对数据更新时，就需要施加写锁了，不在允许其它的操作进行，以免产生数据的脏读和幻读。锁同样有粒度大小，有表级锁(table lock)和行级锁(row lock)，分别在数据操作的过程中完成行的锁定和表的锁定。这些根据不同的存储引擎所具有的特性也是不一样的。MySQL大多数事务型的存储引擎都不只是简单的行级锁，基于性能的考虑，他们一般在行级锁基础上实现了多版本并发控制(MVCC)。这一方案也被Oracle等主流的关系数据库采用。它是通过保存数据中某个时间点的快照来实现的，这样就保证了每个事务看到的数据都是一致的。详细的实现原理可以参考《高性能MySQL》第三版。

**3 事务**

（1）简单的说事务就是一组原子性的SQL语句。可以将这组语句理解成一个工作单元，要么全部执行要么都不执行。默认MySQL中自动提交时开启的（start transaction）

（2）事务具有ACID的特性：

@原子性：事务中的所有操作要么全部提交成功，要么全部失败回滚比如你从取款机取钱,这个事务可以分成两个步骤:1划卡,2出钱.不可能划了卡,而钱却没出来.这两步必须同时完成.要么就不完成.

@一致性： 数据库总是从给一个一致性的状态转换到另一个一致性的状态例如,完整性约束了a+b=10,一个事务改变了a,那么b也应该随之改变.不管数据怎么改变。一定是符合约束

@隔离性：一个事务所做的修改在提交之前对其它事务是不可见的 两个以上的事务不会出现交错执行的状态.因为这样可能会导致数据不一致.

@持久性： 一旦事务提交，其所做的修改便会永久保存在数据库中。

（3）事务的隔离级别：

READ UNCOMMITTED(读未提交)：事务中的修改即使未提交也是对其它事务可见

READ COMMITTED(读提交)：事务提交后所做的修改才会被另一个事务看见，可能产生一个事务中两次查询的结果不同。

REPEATABLE READ(可重读)：只有当前事务提交才能看见另一个事务的修改结果。解决了一个事务中两次查询的结果不同的问题。

SERIALIZABLE(串行化)：只有一个事务提交之后才会执行另一个事务。

（4）死锁：

两个或多个事务在同一资源上相互占用并请求锁定对方占用的资源，从而导致恶性循环的现象。对于死锁的处理：MySQL的部分存储引擎能够检测到死锁的循环依赖并产生相应的错误。InnoDB引擎解决的死锁的方案是将持有最少写锁的事务进行回滚。为了提供回滚或者撤销未提交的变化的能力，许多数据源采用日志机制。例如：sql server使用一个预写事务日志，在将数据应用于（或提交到）实际数据页面前，先写在事务日志上。但是，其他一些数据源不是关系型数据库管理系统，他们管理未提交事务的方式完全不同。只要事务回滚时，数据源可以撤销所有未提交的改变，那么这种技术可用于事务管理。

**4 MySQL存储引擎及应用方案：**

**（1）MySQL采用插件式的存储引擎的架构**，可以根据不同的需求为不同的表设置不同的存储引擎。如： usem

**（2）相关字段介绍：**

Name：显示的是表名

Engine：显示存储引擎，该表存储引擎为MyISAM

Row\_format：显示行格式，对于MyISAM有Dynamic、Fixed和Compressed三种。非别表示表中有可变的数据类型，表中数据类型为固定的，以及表是压缩表的环境。

Rows：显示表中行数

Avg\_row\_length：平均行长度（字节）

Data\_length：数据长度（字节）

Max\_data\_length：最大存储数据长度（字节）

Data\_free：已分配但未使用的空间，包括删除数据空余出来的空间

Auto\_increment：下一个插入行自动增长字段的值

Create\_time：表的创建时间

Update\_time：表数据的最后修改时间

Collation：表的默认字符集及排序规则

Checksum：如果启用，表示整个表的实时校验和

Create\_options：创建表示的一些其它选项

Comment：额外的一些注释信息，根据存储引擎的不同表示的内容也不胫相同。

**（3）常用MySQL存储引擎介绍：**

**@InnoDB引擎：**

将数据存储在表空间中，表空间由一系列的数据文件组成，由InnoDb管理

支持每个表的数据和索引存放在单独文件中(innodb\_file\_per\_table)；

支持事务，采用MVCC来控制并发，并实现标准的4个事务隔离级别，支持外键。

索引基于聚簇索引建立，对主键查询有较高性能。

数据文件的平台无关性，支持数据在不同的架构平台移植

能够通过一些工具支持真正的热备，如XtraBackup等；

内部进行自身优化如采取可预测性预读，能够自动在内存中创建bash索引等

**@MyISAM引擎：**

MySQL5.1默认，不支持事务和行级锁

提供大量的特性如全文索引、空间函数、压缩、延迟更新等

数据库故障后，安全恢复性

对于只读数据可以忍受故障恢复，MyISAM依然非常适用

日志服务器的场景也比较适用，只需插入和数据读取操作

不支持单表一个文件，会将所有的数据和索引内容分别存放在两个文件中

MyISAM对整张表加锁而不是对行，所以不适用写操作比较多的场景

支持索引缓存不支持数据缓存

## 二十三、Mycat概述

**1 mycat是一个数据库中间件，也可以理解为是数据库代理。**在架构体系中是位于数据库和应用层之间的一个组件，并且对于应用层是透明的，即数据库感受不到mycat的存在，认为是直接连接的mysql数据库（实际上是连接的mycat,mycat实现了mysql的原生协议）

**2 mycat的三大功能：分表、读写分离、主从切换；**

**3 分表**

（1）对于数据量很大的表（千万级以上），mysql性能会有很大下降，因此尽量控制在每张表的大小在百万级别。对于数据量很大的一张表，可以考虑将这些记录按照一定的规则放到不同的数据库里面。这样每个数据库的数据量不是太大，性能也不会有太大损失。

（2）mycat自动会帮助我们实现分表功能，而对于应用层来说是透明的，即跟一张表没有什么区别！

（3）mycat分表的实现：首先在mycat的scheme.xml中配置逻辑表，并且在配置中说明此表在哪几个物理库上。此逻辑表的名字与真实数据库中的名字一致！然后需要配置分片规则，即按照什么逻辑分库！分片规则有很多，选取以下几个简单说明：

　　1、根据数据库某字段的hash值片

　　2、截取某字段的几位数字，匹配分区号

　　3、按照时间（年份分表）

分表的原则是尽量避免跨库操作操作，跨库操作会损失很多性能，mycat会对各个库的结果集进行合并，另外就是要考虑扩展之后，尽量使最少量的数据迁移。分表规则很多，很灵活，并且在源码基础上修改分片规则也很容易！

**4 分表之后有什么弊端呢？**

（1）使用存储过程/函数就不太方便了，mycat本身不支持存储过程，是通过注解的方式实现存储过程的调用。并且在所有库上都执行，将执行结果合并，并返回（mysql存储过程返回结果最好使用select方式返回）。分表之后还有一个问题，就是自增ID的问题，在分库场景下，如果使用mysql的自增主键，会导致各个库中主键之间有重复。mycat使用内部的全局序列号解决这个问题；即插入数据的时候，mycat会自动的显示的插入自增主键，使用的是全局序列号；

需要在配置逻辑表的时候，配置上自增字段，自增属性=true 这两个属性；

<table name="fh\_fullnote" primaryKey="FH\_FULLNOTE\_ID" autoIncrement="true" dataNode="dn1,dn2,dn3" rule="sharding-by-substring" />

（2）那么全局序列号存储在什么地方呢？mycat是怎么去取的呢？

　　全局序列号有两种存储方式：基于本地文件和基于数据库的存储；

　　我们是采用基于数据库的存储方式，需要在数据库中建立 mycat\_sequence表

Create Table: CREATE TABLE `mycat\_sequence` (

`name` varchar(50) NOT NULL,

`current\_value` int(11) NOT NULL,

`increment` int(11) NOT NULL DEFAULT '100',

PRIMARY KEY (`name`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8

（3）全局序列号就是存储在这个表中；第一个是全局序列号的名，第二个是当前值，第三个是增量；即每次mycat会取出一批序列号（增量就是作用与此），用完之后再取；如果每次都取，在效率上会有损失。

**5 mycat取全局序列号是通过函数来取的，因此需要在数据库上增加如下三个函数**：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*获取函数当前值\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

DELIMITER $$

USE `mycat1`$$

DROP FUNCTION IF EXISTS `mycat\_seq\_currval`$$

CREATE DEFINER=`root`@`%` FUNCTION `mycat\_seq\_currval`(seq\_name VARCHAR(50)) RETURNS VARCHAR(64) CHARSET utf8

DETERMINISTIC

BEGIN

DECLARE retval VARCHAR(64);

SET retval="-999999999,null";

SET retval="-999999999,null";

SELECT CONCAT(CAST(current\_value AS CHAR),",",CAST(increment AS CHAR)) INTO retval FROM MYCAT\_SEQUENCE

WHERE NAME = seq\_name;

RETURN retval;

END$$

DELIMITER ;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*获取下一批值\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

DELIMITER $$

USE `mycat1`$$

DROP FUNCTION IF EXISTS `mycat\_seq\_nextval`$$

CREATE DEFINER=`root`@`%` FUNCTION `mycat\_seq\_nextval`(seq\_name VARCHAR(50)) RETURNS VARCHAR(64) CHARSET utf8

BEGIN

UPDATE MYCAT\_SEQUENCE

SET current\_value = current\_value + increment WHERE NAME = seq\_name;

RETURN mycat\_seq\_currval(seq\_name);

END$$

DELIMITER ;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*设定值\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

DELIMITER $$

USE `mycat1`$$

DROP FUNCTION IF EXISTS `mycat\_seq\_setval`$$

CREATE DEFINER=`root`@`%` FUNCTION `mycat\_seq\_setval`(seq\_name VARCHAR(50),VALUE INTEGER) RETURNS VARCHAR(64) CHARSET utf8

BEGIN

UPDATE MYCAT\_SEQUENCE

SET current\_value = VALUE

WHERE NAME = seq\_name;

RETURN mycat\_seq\_currval(seq\_name);

END$$

DELIMITER ;

一般来说，每一个表应该对应一个全局序列号；

此对应关系在sequence\_db\_conf.properties配置；

表名 = 全局序列号数据库（dn1）

**6 读写分离**

（1）经过统计发现，对数据库的大量操作是读操作，一般占到所有操作70%以上。所以做读写分离还是很有必要的，如果不做读写分离，那么从库也是一种很大的浪费。

（2）mycat的读写分离也是在scheme.xml里面配置的。配置方式如下:

<writeHost host="hostM1" url="192.168.91.231:3306" user="root" password="123456">

<!-- can have multi read hosts -->

<readHost host="hostS2" url="192.168.91.232:3306" user="root" password="123456" />

</writeHost>

值得注意的是，读库是附属于写库的，如果写库挂掉之后，读库也就不能使用了。

读写分离一般涉及到两个问题：一个是读操作的均衡，是全部走读库？还是部分走读库，部分读写库？ 另一个问题，如果同步不及时或者同步出错，或者实时性要求较高，这种场景下，如何强制走写库？首先来看第一个问题，读写均衡问题：scheme.xml配置文件 dataHost标签中有一个balance属性，该属性的不同值表示不同的含义：

## 二十四、Mysql比较Oracel

**1、锁差异**

• Oracle锁加在数据块上

• InnoDB 是在索引上加锁，所以MySQL锁的粒度没有Oracle 精细。

**2、导入导出**

• Oracle采用EXP /IMP ，EXPDP/IMPDP导入导出。

• MySQL采用mysqldump导出，导入可以采用管道或source。

**3、commit**

• Oracle默认手动提交

• MySQL默认自动提交

**4、SQL 缓存**

MySQL只能缓存结果集，不能缓存SQL解析结果

**5、数据库对象**

• Oracle将数据库对象编译存储，直接执行二进制码

• MySQL只存储代码，临时解析执行，所以MySQL触发器、存储过程、函数等对象创建时仅检查语法，不检查逻辑

**6、事务**

• MySQL不是所有引擎都支持，建议优先使用InnoDB，相比其他引擎有更好的并发性

**7、空字符串处理**

• 而Oracle 只有NULL的概念

• MySQL有空字符串 '' 和NULL，空串不等于NULL

**8、结果集行数限制**

• Oracle使用rownum

• MySQL使用limit;

**9、执行SQL**

• Oracle需要select func() from dual;

• MySQL可以直接select func();

**10、修改字段类型或长度**

• Oracle：alter table tablename modify column ...;

• MySQL：alter table tablename modify (...);

**11、复制数据**

• Oracle的as是必选，create table table1 as select \* from table2;

• MySQL的as是可选，create table table1 select \* from table2;

**12、字符串处理**

• Oracle里用单引号包起字符串

• MySQL用双引号包起字符串

## 二十五、Mysql基础规范

**一、基础规范**

1.【强制】使用InnoDB存储引擎

解读：InnoDB存储引擎是MySQL默认存储引擎，支持事务和行级锁，并发性能更好，CPU及内存缓存页优化使得资源利用率更高

2.【强制】使用utf8字符集，如果有字段需要存储emoji表情之类的，则需要将字段或表设置成utf8mb4

解读：万国码，无需转码，无乱码风险，节省空间，utf8mb4向下兼容utf8

3.【强制】数据表、数据字段必须加入中文注释

解读：便于识别表和字段的用途

4.【强制】禁止使用存储过程、视图、触发器、Event

解读：高并发大数据的互联网业务，架构设计思路是“解放数据库CPU，将计算转移到服务层”，并发量大的情况下，这些功能很可能将数据库拖死，业务逻辑放到服务层具备更好的扩展性，能够轻易实现“增机器就加性能”。数据库擅长存储与索引，CPU计算还是上移吧

5.【强制】禁止存储大文件或者大照片

解读：为何要让数据库做它不擅长的事情？大文件和照片存储在文件系统，数据库里存URI多好

**二、命名规范**

1.【强制】库名、表名、字段名：小写字母，下划线风格，禁止数字开头，禁止两个下划线中间只出现数字，禁止复数名词。

解读：正例：getter\_admin，task\_config，level3\_name 反例：GetterAdmin，taskConfig，level\_3\_name

2.【强制】命名中不允许出现MYSQL数据库中的保留字。如desc、range、match、delayed等，请参考MySQL官方保留字。

3.【强制】索引命名格式为：索引类型\_字段名。普通索引名idx\_xxx，唯一索引名uniq\_xxx

**三、表设计规范**

1.【强制】单实例表数目必须小于500，单表列数目必须小于30

2.【建议】无特殊情况下，建议显式指定一个无业务用途的自增unsigned bigint型主键

解读：主键递增，数据行写入可以提高插入性能，可以避免page分裂，减少表碎片提升空间和内存的使用

3.【建议】表必须有create\_time和update\_time两个字段，并指定datetime类型

解读：

建表时建议按照以下格式设计：

`create\_time` datetime NOT NULL DEFAULT CURRENT\_TIMESTAMP;

`update\_time` datetime NOT NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP;

加字段时建议按照以下格式设计：

ADD COLUMN `create\_time` datetime NOT NULL DEFAULT '0000-00-00 00:00:00';

ADD COLUMN `update\_time` datetime NOT NULL DEFAULT '0000-00-00 00:00:00' ON UPDATE CURRENT\_TIMESTAMP;

4.

timestamp的使用有以下的限制，所以建议统一使用datetime类型

5.

a) timestamp支持的时间范围小，范围是从’1970-01-01 00:00:01′ 到 ’2038-01-19 03:14:07′ UTC；

b) timestamp的插入和查询受当地时区的影响；

c) timestamp的属性会受到MySQL版本和SQL mode的影响。

6.

7.【强制】禁止使用外键，如果有外键完整性约束，需要应用程序控制

解读：外键会导致表与表之间耦合，update与delete操作都会涉及相关联的表，十分影响sql的性能，甚至会造成死锁。高并发情况下容易造成数据库性能，大数据高并发业务场景数据库使用以性能优先

8.【建议】单表行数超过500万行或者单表容量超过2GB，才推荐进行分库分表。

解读：如果预计三年后的数据量根本达不到这个级别，请不要在创建表时就分库分表。

**四、字段设计规范**

1.【强制】把字段定义为NOT NULL并且提供默认值

解读：

以下只是建议的默认值，可根据业务需要设置成其它默认值

tinyint/int/bigint 类型默认值：0

char/varchar 类型默认值：' '

date 类型默认值：'0000-00-00'

time 类型默认值：'00:00:00'

datetime 类型默认值：'0000-00-00 00:00:00'

2.

a）null的列使索引/索引统计/值比较都更加复杂，对MySQL来说更难优化

b）null 这种类型MySQL内部需要进行特殊处理，增加数据库处理记录的复杂性；同等条件下，表中有较多空字段的时候，数据库的处理性能会降低很多

c）null值需要更多的存储空间，无论是表还是索引中每行中的null的列都需要额外的空间来标识

d）对null 的处理时候，只能采用is null或is not null，而不能采用=、in、<、<>、!=、not in这些操作符号。如：where name!=’shenjian’，如果存在name为null值的记录，查询结果就不会包含name为null值的记录

3.【建议】禁止使用TEXT、BLOB类型，可以做垂直拆分到子表中

解读：会浪费更多的磁盘和内存空间，非必要的大量的大字段查询会淘汰掉热数据，导致内存命中率急剧降低，影响数据库性能

4.【强制】存储货币用decimal或整数类型，禁止使用float和double

解读：float和double在存储的时候，存在精度损失的问题

5【建议】小数类型用decimal，禁止使用float和double

解读：float和double在存储的时候，存在精度损失的问题，很可能在值的比较时，得到不正确结果。如果存储的数据范围超过decimal的范围，建议将数据拆成整数和小数分开存储。

6.【建议】枚举类型禁止使用ENUM，可使用TINYINT代替

解读：

a）增加新的ENUM值要做DDL操作

b）ENUM的内部实际存储就是整数，你以为自己定义的是字符串？

7 【建议】表达是与否概念的字段，必须使用is\_xxx的方式命名，数据类型是unsigned tinyint（ 1表示是，0表示否）

8 【建议】使用varchar(20)存储手机号

解读：

a）涉及到区号或者国家代号，可能出现+-()

b）手机号会去做数学运算么？

c）varchar可以支持模糊查询，例如：like“138%”

1. 【建议】如果存储的字符串长度几乎相等，使用char定长字符串类型。

10.【建议】varchar是可变长字符串，不预先分配存储空间，长度不要超过5000。

11.【建议】字段允许适当冗余，以提高性能，但是必须考虑数据同步的情况。

解读：冗余字段应遵循：不是频繁修改的字段，不是varchar超长字段，更不能是text字段。正例：商品类目名称使用频率高，字段长度短，名称基本一成不变，可在相关联的表中冗余存储类目名称，避免关联查询。

12.【建议】选择合适的字符存储长度，不但节约数据库表空间、节约索引存储，更重要的是提升检索速度。

**五、索引设计规范**

1.【建议】单表索引建议控制在5个以内

解读：索引过多会增加存储开销和增删改的开销

2.【建议】禁止在更新十分频繁、区分度不高的属性上建立索引

解读：

a）更新会变更B+树，更新频繁的字段建立索引会大大降低数据库性能

b）“性别”这种区分度不大的属性，建立索引是没有什么意义的，不能有效过滤数据，性能与全表扫描类似

3.【建议】尽量使用组合索引，建立组合索引时必须把区分度高的字段放在前面，字段数不允许超过5个

解读：能够更加有效的过滤数据，索引上字段超过5个时，实际已经起不到有效过滤数据的作用了

1. 【建议】在排序、分组、取唯一字段上创建索引，经常与其他表进行关联的表，在关联字段上应该建立索引，经常出现在Where子句中的字段，特别是大表的字段，应该建立索引

5.【建议】业务上具有唯一特性的字段，即使是组合字段，也必须建成唯一索引。

解读：不要以为唯一索引影响了insert速度，这个速度损耗可以忽略，但提高查找速度是明显的；另外，即使在应用层做了非常完善的校验和控制，只要没有唯一索引，根据墨菲定律，必然有脏数据产生。

6.【建议】在varchar字段上建立索引时，必须指定索引长度，没必要对全字段建立索引，根据实际文本区分度决定索引长度。

解读：索引的长度与区分度是一对矛盾体，一般对字符串类型数据，长度为20的索引，区分度会高达90%以上，可以使用count(distinct left(列名, 索引长度))/count(\*)的区分度来确定。

**六、SQL使用规范**

1.【强制】禁用select \*，必须指定列

解读：

a）读取不需要的列会增加CPU、IO、NET消耗

b）不能有效的利用覆盖索引

c）使用SELECT \*容易在增加或者删除字段后出现程序BUG

2.【强制】insert语句需指定列，INSERT INTO table\_name (列1, 列2,...) VALUES (值1, 值2,....)

解读：避免在增加或者删除字段后出现程序BUG

3.【强制】禁止使用属性隐式转换

解读：SELECT uid FROM t\_user WHERE phone=13812345678 会导致全表扫描，而不能命中phone索引，因为phone是varchar类型，需要添加引号

4.【建议】禁止在WHERE条件的属性上使用函数或者表达式

解读：SELECT uid FROM t\_user WHERE from\_unixtime(day)>='2017-02-15' 会导致全表扫描

正确的写法是：SELECT uid FROM t\_user WHERE day>= unix\_timestamp('2017-02-15 00:00:00')

5.【强制】禁止负向查询，以及%开头的模糊查询

解读：

a）负向查询条件：NOT、!=、<>、!<、!>、NOT IN、NOT LIKE等，会导致全表扫描

b）%开头的模糊查询，会导致全表扫描

6.【建议】禁止大表使用JOIN查询，禁止大表使用子查询

解读：会产生临时表，消耗较多内存与CPU，极大影响数据库性能

7.【建议】禁止使用OR条件，必须改为IN查询

解读：旧版本Mysql的OR查询是不能命中索引的，即使能命中索引，为何要让数据库耗费更多的CPU帮助实施查询优化呢？

1. 【强制】应用程序必须捕获SQL异常，并有相应处理

9.【建议】超过三个表禁止join。需要join的字段，数据类型保持绝对一致；多表关联查询时，保证被关联的字段需要有索引。

解读：即使双表join也要注意表索引、SQL性能。

10.【强制】页面搜索严禁左模糊或者全模糊，如果需要请走搜索引擎来解决。

解读：索引文件具有B-Tree的最左前缀匹配特性，如果左边的值未确定，那么无法使用此索引。

11.【建议】使用order by请注意利用索引的有序性。

解读：order by 最后的字段是组合索引的一部分，并且放在索引组合顺序的最后，避免出现file\_sort的情况，影响查询性能。

正例：where a=? and b=? order by c; 索引：a\_b\_c

反例：索引中有范围查找，那么索引有序性无法利用，如：WHERE a>10 ORDER BY b; 索引a\_b无法排序。

12.【建议】利用覆盖索引来进行查询操作，来避免回表操作。

解读：如果一本书需要知道第11章是什么标题，会翻开第11章对应的那一页吗？目录浏览一下就好，这个目录就是起到覆盖索引的作用。 正例：能够建立索引的种类：主键索引、唯一索引、普通索引，而覆盖索引是一种查询的一种效果，用explain的结果，extra列会出现：using index。

13.【建议】利用延迟关联或者子查询优化超多分页场景。

解读：MySQL并不是跳过offset行，而是取offset+N行，然后返回放弃前offset行，返回N行，那当offset特别大的时候，效率就非常的低下，要么控制返回的总页数，要么对超过特定阈值的页数进行SQL改写。 正例：先快速定位需要获取的id段，然后再关联： SELECT a.\* FROM 表1 a, (select id from 表1 where 条件 LIMIT 100000,20 ) b where a.id=b.id

14.【建议】SQL性能优化的目标：至少要达到 range 级别，要求是ref级别，如果可以是consts最好。

解读：

1）consts 单表中最多只有一个匹配行（主键或者唯一索引），在优化阶段即可读取到数据。 2）ref 指的是使用普通的索引（normal index）。

3）range 对索引进行范围检索。

反例：explain表的结果，type=index，索引物理文件全扫描，速度非常慢，这个index级别比较range还低，与全表扫描是小巫见大巫。